NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

UPGRADABLE OPERATIONAL AVAILABILITY FORECASTING TOOL FOR THE U.S. NAVY P-3 REPLACEMENT AIRCRAFT

by

Michael C. Margolis

September 2003

Thesis Advisor:	Arnold H. Buss
Second Reader:	David A. Schrady

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Upgradeable Operational Availability Forecasting Tool for the U.S. Navy P-3 Replacement Aircraft			**5. FUNDING NUMBERS**
6. AUTHOR(S) Michael C. Margolis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The P-3 Orion maritime aircraft has been the U.S. Navy's primary maritime patrol aircraft since its fleet introduction in 1962. Naval Aviation Systems Command (NAVAIR) has determined that the P-3 fleet has sufficiently aged to warrant a replacement. The replacement aircraft is currently undergoing the conceptual phase of development and it is during this period that NAVAIR is interested in evaluating the trade-off between operational availability and the associated cost to achieve this operational availability. This thesis developed a simulation tool that was used to investigate relationships that affect cost and operational availability of the new (notional) aircraft on a deployment. The simulation tool was exercised for select scenarios in order to gain insights into the value of investing funds in additional aircraft versus the value of investing funds in increased component reliability. The simulation was developed to be very flexible and extensible, enhancing its value for future analyses. Required data inputs into the simulation tool are formatted utilizing a new technology called Extensible Markup Language (XML) which facilitates use of the data in nearly all computer software packages. The model is robust in nature and can be applied to a wide variety of aircraft.

14. SUBJECT TERMS Trade-Off Analysis, Operational Availability, Readiness Based Sparing, Cost Analysis, P-3 Orion			**15. NUMBER OF PAGES** 130
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

UPGRADABLE OPERATIONAL AVAILABILITY FORECASTING TOOL FOR THE U.S. NAVY P-3 REPLACEMENT AIRCRAFT

Michael C. Margolis
Captain, United States Marine Corps
B.S., Mary Washington College, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 2003

Author: Michael C. Margolis

Approved by: Arnold H. Buss
Thesis Advisor

David A. Schrady
Second Reader

James N. Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The P-3 Orion maritime aircraft has been the U.S. Navy's primary maritime patrol aircraft since its fleet introduction in 1962. Naval Aviation Systems Command (NAVAIR) has determined that the P-3 fleet has sufficiently aged to warrant a replacement. The replacement aircraft is currently undergoing the conceptual phase of development and it is during this period that NAVAIR is interested in evaluating the trade-off between operational availability and the associated cost to achieve this operational availability. This thesis developed a simulation tool that was used to investigate relationships that affect cost and operational availability of the new (notional) aircraft on a deployment. The simulation tool was exercised for select scenarios in order to gain insights into the value of investing funds in additional aircraft versus the value of investing funds in increased component reliability. The simulation was developed to be very flexible and extensible, enhancing its value for future analyses. Required data inputs into the simulation tool are formatted utilizing a new technology called Extensible Markup Language (XML) which facilitates use of the data in nearly all computer software packages. The model is robust in nature and can be applied to a wide variety of aircraft.

THIS PAGE INTENTIONALLY LEFT BLANK

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs and data herein are free of computational, logic, and collection errors, they cannot be considered validated. Any application of these programs or data without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge those individuals who provided their support:

To Dr. Arnold Buss. Thank you for your support, encouragement, and guidance. It is an honor and privilege having you as my advisor!

To Dr. David Schrady. Thank you for keeping me honest and encouraging me to put words on paper early on.

To Bill Kroshl and Jack Keane. Thank you for giving me the opportunity to conduct my thesis experience tour at Johns Hopkins University. My time at the Applied Physics Lab definitely put the "experience" in the "experience tour."

To my Mom, Dad, and sister Julie. Thank you for supporting my education over the years. I would not have had the opportunity to come to NPS if it was not for your support.

To MAJ Stephanie Tutton, MAJ Joe Baird, Capt Wolfgang Lehmann, and Capt Eric Wolf. Thank you for being my study group and most importantly my friends.

To all my classmates. Each of you helped me at one point or another on this thesis and I am very thankful your assistance. I am truly blessed and honored to have had the opportunity of serving with each and everyone of you and I look forward to serving with you again in the future.

I am especially grateful to Julie Webster. Thank you for your love, support and encouragement throughout this process. I love you.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The replacement aircraft for the U.S. Navy's P-3 Orion aircraft is currently in the conceptual phase of development and Naval Air Systems Command (NAVAIR) is interested in evaluating the trade-off between cost and operational availability of the conceptual design(s).

This thesis creates a simulation called "SIMACE" that was used to investigate various relationships between the operational availability and cost of the new (notional) aircraft on a deployment. SIMACE models the activity of a notional squadron of aircraft and utilizes software named "SIMKIT," a Java programming language package used to develop discrete event simulations. The simulation was developed to be flexible and extensible. New modules as well as different types of aircraft can easily be represented in the model. Specific results should be seen as representational of the kinds of analysis that can be done with SIMACE.

The simulation uses data input from a new technology called Extensible Markup Language (XML) documents. The XML documents created for this thesis enable the user to apply the same data to nearly any programming language with little or no modification. The use of XML allows the data to be easily applied to future software programs.

Analysis suggests that the choice of distributions used to represent aircraft component time to failures (random variates) can have a significant impact on generated operational availability estimates. The various databases used to obtain data for this thesis provide only point estimates representing mean values, which limits what random variates can be created and utilized. This thesis demonstrates that the assumption about which distribution to use to create component time to failure random variates is critical and is a strong reason why data collection systems should provide more information than just the mean.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The P-3 Orion maritime aircraft has been on the cutting edge of maritime patrol since its fleet introduction in 1962. The first thirty years of the P-3's existence was spent accomplishing one of the United States' most important missions, tracking the Soviet Union's submarine fleet. After decades of mission and aircraft updates, Naval Aviation Systems Command (NAVAIR) has determined that the P-3 fleet has sufficiently aged to warrant a replacement. Deteriorating material condition and obsolescence issues have contributed to reduced P-3 availability. The first fleet P-3 will reach its Fatigue Life Expended (FLE) limit in 2004. By 2007 approximately 40 aircraft will reach this level. By 2011, 83 P-3 aircraft will reach FLE, which will place the maritime community below their required inventory of 198 aircraft (Elward, 2000).

The P-3 fleet will be replaced and the purpose of the replacement aircraft is to provide a weapon system to recapitalize on the P-3's capability (Elward, 2000). Although the design of this new aircraft has not been determined, NAVAIR wishes to investigate various factors that may affect this notional aircraft's operational availability.

One element of forecasting the overall operational availability of the new type/model/series of aircraft is forecasting the availability of such aircraft on a deployment. The purpose of this thesis is to create a simulation that can be used to investigate relationships that affect cost and operational availability of the new (notional) aircraft described above on a deployment. The simulation was developed to be flexible and extensible. New modules as well as different types of aircraft can easily be represented in the model.

B. BACKGROUND

There are three levels of Naval aviation maintenance: Organizational, Intermediate, and Depot. Organizational Level (O-level) maintenance is performed by an operating unit in support of its own operations and is usually accomplished by maintenance personnel assigned to the operating unit. The O-level maintenance mission

is to maintain assigned aircraft in a fully mission capable status. The purpose of the Intermediate Level (I-level) of maintenance is to enhance and sustain the combat readiness and mission capability of supported activities by providing quality and timely material support at the nearest location with the lowest practical resource expenditure. I-level maintenance includes, but is not limited to the repair of individual components removed from the aircraft. For example, an individual component breaks at the O-Level and is subsequently delivered to the I-Level for repair or replacement. The third level of Naval aviation maintenance is the Depot Level (D-level) and its purpose is to ensure continued flying integrity of airframes and flight systems during subsequent service periods. D-level maintenance performs major overhaul or rebuilding of parts, assemblies, and subassemblies. It also supports O-level and I-level maintenance by providing engineering assistance and performing maintenance that is beyond their capabilities (OPNAVINST 4790.2H, 2001).

Operational availability (A_o) represents the expected percentage of time that a weapons system or individual equipment is ready to perform satisfactorily in an operating environment and is comprised of three main factors: reliability, maintainability, and supportability. A measure of reliability is the time between failures. The time span between subsequent failures of a particular component is a measure of how reliable the component is. An example of maintainability is the time necessary to remove and replace a broken component and has much to do with the number of available maintenance personnel, their respective skill levels, and the design of the unit or system. Supportability is measured by the expected response time for logistic support and administrative delay (OPNAVINST 3000.12, 1987). For example, how long it takes to order and receive a new component. A more detailed explanation of A_o and its associated factors is in Appendix B (Calculation of Operational Availability).

Similar to a squadron of P-3, a squadron of the new (notional) aircraft will deploy on a regular basis for a specified period of time. While the notional squadron may consist of nine aircraft, four of the nine aircraft may operate from a satellite location away from the remaining five aircraft for a portion of the deployable period. The aircraft and maintenance personnel operating from this satellite location is called a “detachment.”

All aircraft squadrons have the capability to perform O-level maintenance. Although the notional squadron usually operates from locations (sites) providing I-level maintenance, this is not always the case. The entire squadron may initially deploy to a site possessing I-level support, however the detachment(s) may not have this support available. Thus, the detachment has the difficult task of maintaining a level of readiness necessary to accomplish assigned missions without the added support of the I-level. To maintain a sufficient level of A_o the detachment commander must ensure the proper quantity of maintenance personnel and spare parts are embarked in order to quickly repair aircraft components when they fail. Aircraft parts are very expensive and available in limited quantities. Therefore, the minimum quantity of spare parts necessary to achieve a target or goal A_o must be calculated and embarked for the detachment. This minimum quantity of spares is called a "spares kit" and is the minimum expected supply expenditure necessary to achieve a particular A_o .

Like any Naval aircraft, the notional aircraft consists of components called Weapon Replaceable Assemblies (WRAs). The official definition of a WRA, per OPNAVINST 4790.2H, is a generic term referring to all the replaceable packages of an avionic equipment, pod, or system as installed in an aircraft weapon system, with the exception of cables, mounts, and fuse boxes or circuit breakers (OPNAVINST 4790.2H, 2001). However, the term WRA is most commonly used to refer to all repairable components that can be removed and replaced on an aircraft. This thesis uses data on 449 WRAs to represent the notional aircraft. That is, each aircraft in this thesis consists of 449 WRAs. The list of WRAs which is used to represent the notional aircraft was obtained from NAVAIR and Naval Inventory Control Point (NAVICP) Philadelphia.

"Fully mission capable" (FMC) refers to the material condition of an aircraft such that it can perform all of its missions. "Mission Capable" (MC) refers to the material condition of an aircraft that can perform at least one of its missions. An aircraft can be FMC only when all installed WRAs on the aircraft are 100 percent operational. This thesis conducts a limited analysis of A_o as it pertains to operationally available aircraft,

budget for spares, WRA reliability, and the number of aircraft on a detachment. There are several other factors worthy of analysis and this thesis serves as a springboard for such inquiry.

C. METHODOLOGY

In 1998 the Department of Defense (DoD) decreed that Readiness-Based Sparing (RBS) shall be used for weapon system support provisioning requirements computations so that the resulting investment in supplies will meet end item readiness objectives at minimum cost (DODINST 4140.1-R, 1998). To examine relationships between A_o, cost of spare WRAs, number of aircraft, and WRA reliability for a detachment of the notional aircraft, it is first necessary to determine minimum cost spares kits based on the principles of RBS. After the spares kit is determined, a simulation model is used to better analyze the relationships of interest. The spares kit that determines a particular A_o can be viewed as a function of the available budget.

A model called the Personal Computer (PC) Aviation Retail Requirements Oriented to Weapon Replaceable Assemblies (ARROWs) was used to estimate the minimum cost spares kits used in this thesis. PC ARROWs is a RBS model for developing consumer level inventory requirements. The model computes and evaluates readiness-based spares and repair part requirements in support of aviation weapon systems (Burrows, 1994). It produces a list of spare parts necessary to keep a group of weapon systems up and running and ready to use. That is, given a target A_o goal, PC ARROWs produces a spares kit of WRAs to achieve the desired level of A_o. After the spares kits are determined, another model called "SIMACE" is used to better approximate what A_o could be achieved by the inventory produced by PC ARROWs. SIMACE is a discrete event simulation developed by the author of this thesis. The inventory produced by PC ARROWs is used as input into SIMACE and is the expected minimum budget necessary to achieve the A_o simulated by SIMACE.

D. ORGANIZATION OF STUDY

Chapter II explains how the data used in this thesis was collected and how the data is managed. Chapter III describes the simulation model SIMACE as well as the scenario and experimental design. Chapter IV presents the results and analysis, and Chapter V the conclusions and recommendations. Appendix A provides a glossary of acronyms, Appendix B provides an overview on the calculation of A_o, Appendix C displays the simulation output, and Appendix D provides a copy of the SIMACE source code.

THIS PAGE INTENTIONALLY LEFT BLANK

II. OVERVIEW OF DATA

A. DATA RESOURCES

A list of WRA National Item Identification Numbers (NIINs), which this thesis uses to comprise the notional aircraft, was obtained from NAVAIR and Naval Inventory Control Point (NAVICP) Philadelphia and is a collection of pre-existing P-3 and EP-3 components. The collected list of WRAs is not a true list of components that comprise the replacement aircraft for the P-3 because the replacement aircraft does not exist at the time of the writing of this thesis. Data is notional and is only used to illustrate the value of the simulation tool developed in this thesis.

Several pieces of data were required for each WRA in addition to a NIIN. All of the following information was collected by the author of this thesis from the Naval Aviation Logistics Data Analysis (NALDA) Logistics Management Decision Support System website (NALDA, 2003):

1) Nomenclature
2) Mean Time Between Failure
3) Mean Time to Repair
4) Unit Cost

In addition to the information above, an estimated quantity of each WRA per aircraft was obtained from NAVAIR and NAVICP.

B. INPUT DATA AS XML DOCUMENT

After the data was collected, it was organized into Extensible Markup Language (XML) documents for input into the simulation tool SIMACE, which is described in Chapter III (Simulation Model Development).

XML technology is used to model data for computer processing. It is platform and language independent, open source, license free, and has international standards. XML technology addresses how to represent data and surrounding information to describe its content and form thereby enhancing the data's meaning. For example, sections in a newspaper are differentiated by their spacing and position on the page and

the use of different fonts for titles and headings. XML works much the same way but uses symbols instead of spaces and fonts (Ray, 2001).

If an input file has no boundaries or labels then a program cannot possibly know how to treat a piece of text and distinguish it from any other piece. A newspaper without spaces and only one font style is a large and uninteresting block of text. A computer program would not be able to distinguish where a particular article began or end. XML solves this problem (Ray, 2001).

Figure 1 displays a portion of one of the XML documents created for this thesis:

```
<WRA>
    <Label>000039137</Label>
    <Nomenclature>INDICATOR DETECTING</Nomenclature>
    <UnitPrice>28372.00</UnitPrice>
    <QtyPerAircraft>1</QtyPerAircraft>
    <QtySpare>2</QtySpare>
    <MTTFDistribution>Exponential</MTTFDistribution>
    <MTTFValue>59678.86</MTTFValue>
    <MTTRDistribution>Exponential</MTTRDistribution>
    <MTTRValue>3.9</MTTRValue>
    <OSTDistribution>Exponential</OSTDistribution>
    <OSTValue>169.0</OSTValue>
</WRA>
```

Figure 1 Sample Portion of XML Document

Note that in Figure 1 there are special symbols called “markup” or “tags.” The tag <Nomenclature> is called a start tag, and the tag </Nomenclature> is called an end tag and they define the beginning and end of a collection of text. That is, they act as bookends marking the beginning and end of data. Each set of “tags” and the data in-between them are collectively called an “element” (Ray, 2001).

XML combines data into hierarchitical structures. The items in 0 relate to each other in parent/child relationships. For example, elements <Nomenclature>, <Label>, <UnitPrice>, and <OSTValue> are all children of the <WRA> element.

Data in Figure 1 are referenced by their element name thus making the querying of data less prone to errors. That is, information formatted according to XML standards is "self-describing" (Hunter, 2001). Looking at the data, the reader can easily locate the information pertaining to "Nomenclature." If the nomenclature text is needed, it is obtained simply by calling the element <Nomenclature> which returns the text "INDICATOR DETECTING." Note that if element <WRA> is called, all elements contained between the tags <WRA> and </WRA> would be returned (the child elements).

XML is not a computer language, but a standard for creating markup languages (i.e. tags, element names) that meet XML criteria. In other words, XML describes a syntax that can be used for individuals to create their own unique markup language. Individuals are able to create their own set of "tags" in which to describe their data when using XML (Hunter, 2001).

Another value to using data stored in an XML document is that it can easily be checked for errors prior to being used in a computer program. To do this, a schema was developed to validate the data prior to its use. A schema is a "template" that sets the data requirements and provides a way to define the XML document. As an XML-based language is used for structuring XML documents, the schema describes constraints that govern the order and sequence of data and specifies permissible value spaces for all data used inside the document (Kim, 2003). Figure 2 displays a selection of schema restrictions used to validate the XML documents used in this thesis:

Element	Restriction
WRA	Can occur multiple times, > 0 and unbounded
Nomenclature	Must occur exactly once within each WRA Element String of any length
QtyPerAircraft	Must occur exactly once within each WRA Element Integer value that is >= 0
UnitPrice	Must occur exactly once within each WRA Element Double value that is >= 0

Figure 2 Sample Schema Restrictions

If a required data element is missing, the schema will not validate the XML document. Something as simple as an extra "space" character in the input file can ruin a simulation run. Not only could an unwanted space character ruin the simulation, for large files it could take hours, if not days, to locate the error. When validating an XML document with a schema, the schema will find the errors for the user thus saving hours (possibly days) of needless troubleshooting.

The XML documents created in this thesis enable the user to apply the same data to nearly any programming language with little or no modification. If the simulation model developed in this thesis becomes obsolete, the data can still be applied to future programs. That is, data are structured in a robust way as to make it readable by simulation models other than the one created for this thesis and described in Chapter III.

III. SIMULATION MODEL DEVELOPMENT

A. OVERVIEW

SIMACE was developed by the author of this thesis, and it is a discrete event simulation tool that models the activity of a notional squadron of aircraft. SIMACE is built on software named "SIMKIT" which is a Java package used to develop discrete event-based simulations (Buss, 2002). SIMACE utilizes a model called PC ARROWs to compute readiness-based spares and repair part requirements for secondary items stocked in support of aviation systems and produces a list of spare parts (a "spares kit") necessary to keep a group of weapon systems up and running and ready to use (Burrows, 1994). The inventory level produced by PC ARROWs is the expected minimum budget necessary to achieve an A_o goal. After the spares kit is determined, SIMACE is used to better approximate what A_o can be achieved by the inventory produced by PC ARROWs.

B. THE SCENARIO

The scenarios presented in this thesis assume the notional detachment of aircraft operate from a deployed site without I-Level support for a period of 90-days. The sortie requirements are similar to that of a search and detection mission where 24-hour surveillance of a particular area is required. Before one aircraft can depart the patrol area, another aircraft must arrive to relieve the aircraft currently on patrol.

C. SIMULATION INPUTS

The model developed in this thesis requires the following input data:

NumAircraft –The number of aircraft on the detachment.

TransitTime – The flight time (hours) from the detachment's base of operations to the patrol area (each-way).

PatrolTime – The flight time (hours) each aircraft is required to be on patrol. That is, the flight time each aircraft must provide search and detection operations over the patrol area.

InspectionTime – The time required to conduct a post-flight inspection. Every time an aircraft returns from a sortie it must undergo a post-flight inspection before becoming available for another sortie.

DeploymentLength – The time length (hours) of the detachment's mission.

MTTFSeed – A seed for the MTTF random variates.

MTTRSeed – A seed for the MTTR random variates.

OSTSeed – A seed for the OST random variates.

A list of WRAs. For each WRA the following data are required:

Label – A unique identifier for the particular type WRA. For example, all widgets must have a label that is unique to widgets. The possibility exists that different type WRAs can have the same nomenclature. However, each type WRA must have a unique identifier (label) such as a part number or NIIN.

Nomenclature – Name of the WRA.

UnitPrice – Cost of a replacement WRA, in dollars.

QtyPerAircraft (Quantity per Aircraft) – The quantity of this type WRA that is installed on each aircraft.

QtySpare (Quantity Spare) – The quantity of this type WRA that is to be included in the spares kit.

MTTFDistribution – The statistical distribution from which times to failure originate. Used to produce a MTTF random variate.

MTTFValue – The parameter (mean value) of the MTTFDistribution. Used to produce a MTTF random variate.

MTTRDistribution – The statistical distribution from which times to repair originate. Used to produce a MTTR random variate.

MTTRValue – The parameter (mean value) of the MTTRDistribution. Used to produce a MTTR random variate.

OSTDistribution – The statistical distribution from which ordering and shipping times originate. Used to produce an OST random variate.

OSTValue –The parameter (mean value) of the OSTDistribution. Used to produce an OST random variate.

D. MODEL ASSUMPTIONS

Most scenarios presented in this thesis assume the Exponential distribution for MTTF, MTTR, and OST due to the lack of data to parameterize other distributions. However, one scenario in this thesis uses the Weibull distribution for MTTF. SIMACE's robust design allows for the application of any probability distribution (not only Exponential) for MTTF, MTTR, and OST, which are described next.

Reliability data (i.e. MTTF) are frequently modeled using a Weibull distribution. Weibull distributions are often used when the rate at which failures occur increases monotonically with the accumulation of service life. For example, the failure rates of a hydraulic pump are believed to increase with continued usage. The longer the hydraulic pump operations, the more likely it is to fail. The Weibull distribution takes wear from usage into consideration whereas the Exponential distribution does not. In other words, the Exponential distribution has a constant hazard function that does not vary with accumulated service. For example, given a hydraulic pump's MTTF is 1000 hours, its expected time to failure (assuming Exponential distribution) is always 1000 hours (Locks, 1973). The reason the Exponential is used rather than the Weibull is because the Weibull distribution requires two parameters whereas the Exponential distribution requires only one (the mean). The only data available at the time of the writing of this thesis is a point estimate for the MTTF, hence the utilization of the Exponential distribution.

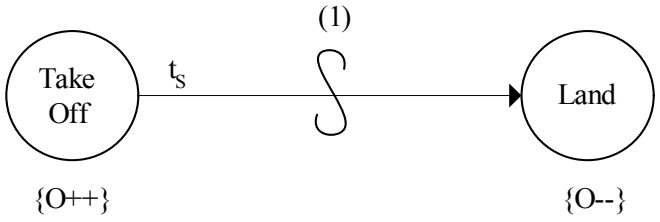
The LogNormal distribution fits data corresponding to maintenance repair times (i.e. MTTR) for complex systems and equipment. It applies to most maintenance tasks and repair actions where task times and frequencies vary (Blanchard, 1998). Ideally, MTTR should be distributed via a LogNormal distribution, but lack of data to parameterize LogNormal distributions necessitates use of the Exponential for the

scenarios considered in this thesis. The reason the Exponential is used rather than the LogNormal is because the LogNormal distribution requires two parameters whereas the Exponential distribution requires only one parameter (the mean). The only data available at the time of the writing of this thesis is a point estimate for the MTTR, hence the utilization of the Exponential distribution.

Supply requisitioning (ordering and receiving replacement WRAs) occurs and the amount of time to complete a requisition is an input parameter. The scenarios presented in this thesis assume OST to follow an Exponential distribution. A better choice for the OST distribution may prove to be the Normal distribution because shipping times often follow a fixed amount of time with little variation (Blanchard, 1998). However, the Exponential distribution is used to generate OST times for the same reason the Exponential distribution is used to generate MTTR times: lack of data to parameterize a two parameter distribution (the Normal distribution in this case).

E. EVENT GRAPH NOTATION

The various processes described in this chapter are presented graphically in "event graphs." Event graphs specify scheduling relationships between events. A simple example of an event graph is presented in Figure 3. Throughout this thesis, simulation entities are displayed in *italics*.



Conditions :

(1) If an *Aircraft* is available for a sortie.

Figure 3 Event Graph Example

Figure 3 says that an *Aircraft* takes-off for a sortie and lands after a time duration t_S if there is an *Aircraft* available for a sortie. {O++} means that when an *Aircraft* takes-off, the number of operating (flying) *Aircraft* is incremented by 1. {O--} means that when an *Aircraft* lands, the number of operating *Aircraft* is decremented by 1. The "S" shape in Figure 3 acts as a "conditional" for the transition to occur. That is, the Land event will be scheduled only if an *Aircraft* is available for a sortie. This is the notational convention followed for event graphs throughout this thesis.

F. SIMULATION METHODOLOGY

1. SIMACE Initialization

As described in Chapter II, SIMACE uses data stored in external XML documents to create software objects representing various entities within the simulation. Using this data, SIMACE instantiates a squadron of *Aircraft* objects. After the squadron is instantiated, objects representing *WRAs* are created. As data pertaining to each *WRA* is read into SIMACE from the XML document, *WRA* objects are instantiated and added to a "master list" from which copies of new *WRA* objects are made. MTTF, MTTR, and OST random variates are created and it is from these random variates that new times to failure (TTF), new times to repair (TTR), and new requisition times (OST) are generated.

Each *Aircraft* is assigned a unique, two-digit tail number (i.e. "00", "01", "12") which is how SIMACE references each individual *Aircraft*. The *Aircraft* are then populated with the proper quantity of each *WRA*. As each *WRA* is installed on an *Aircraft*, the newly installed *WRA* is assigned a TTF generated from the MTTF random variate.

After an *Aircraft* is instantiated and populated with the proper quantity of each *WRA,* the *Aircraft* is added to a queue of "ready" *Aircraft*. As requirements for sorties become known, the *Aircraft* with the least amount of operating time (OT) is selected from the queue of ready *Aircraft* and then tasked for the next sortie. OT is defined as the number of flight hours accumulated by each individual *Aircraft.* Selecting *Aircraft* in this manner most evenly distributes flight time and utilization among all *Aircraft.* If there is a

tie between *Aircraft* for the least amount of OT, the priority goes to the *Aircraft* with the lowest tail number. If future research deems a change to this selection policy is appropriate, SIMACE's robust design allows the user to very easily make this change.

SIMACE calculates A_o by monitoring the quantity of *Aircraft* that are operating (and for how long) and the quantity of *Aircraft* that are on stand-by (and for how long). An *Aircraft* is defined to be operational when it is flying and is defined to be on stand-by when ready for a sortie, but on the ground. From this data, the expected percentage of time an *Aircraft* is available is calculated per (Equation B.7) located in Appendix B (Calculation of Operational Availability).

2. The Sortie Process

SIMACE commences flight operations by immediately tasking an *Aircraft* for take-off. When an *Aircraft* takes-off it notifies all of its *WRAs* that a sortie is starting and adjusts the number of *Aircraft* operating and number of *Aircraft* on stand-by accordingly. Immediately after taking-off, the time at which the *Aircraft* starts its patrol is scheduled. This event is called "StartPatrol". The number of flight hours to fly from the detachment site to the patrol area is called "TransitTime" and is included as an input parameter.

Event StartPatrol schedules an event named "EndPatrol" after a time specified by the input parameter PatrolTime. Event EndPatrol then schedules an event named "Land" to occur after a time determined by the input parameter TransitTime. Event Land tells the *Aircraft* what time to land at the detachment site. However, before the *Aircraft* can depart the patrol area, another *Aircraft* must arrive to the patrol area to ensure continuous coverage. This is handled by scheduling another *Aircraft* to be needed every time an *Aircraft* takes-off on a sortie. For example, if TransitTime is two hours and PatrolTime is eight hours, another *Aircraft* must be ready to relieve the *Aircraft* currently on patrol ten hours after the *Aircraft* on patrol takes-off. Therefore, if available, an *Aircraft* will always take-off "PatrolTime" after the *Aircraft* ahead of it takes-off. If for some reason an *Aircraft* is not available at the time at which an *Aircraft* is needed, the *Aircraft* currently on patrol returns to the detachment site at the specified time therefore leaving the patrol area unattended.

During event Land, the number of operational *Aircraft* is reduced by one to reflect the return of one *Aircraft.* *WRAs* on the *Aircraft* are then notified that the sortie has ended and the amount of time the *Aircraft* operated for (sortie time) is deducted from each *WRA's* TTF. If after the adjustment of a TTF the TTF is found to be negative, the *WRA* is considered failed and is added to the *Aircraft's* list of failed *WRAs*. After all *WRA* TTFs are adjusted, the *Aircraft* is immediately scheduled a post-flight inspection by an event named "StartInspect" which is described in the next section.

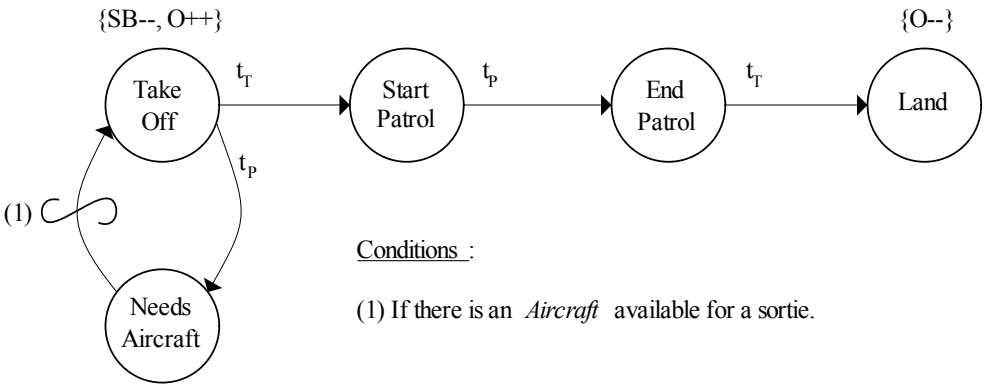


Figure 4 The Sortie Process Event Graph

In Figure 4, t_T is the TransitTime, t_P is the PatrolTime, "SB" is the number of *Aircraft* on stand-by, and "O" is the number of *Aircraft* operating (flying).

3. The Post-Flight Inspection Process

Event StartInspect is scheduled by event Land. Event StartInpect simply schedules an event named "EndInspect" to occur after a time specified by the input parameter "InspectionTime".

The *Aircraft* emerges from event EndInspect in one of two conditions: FMC or Not-Mission Capable (NMC). The *Aircraft* is FMC if its list of failed *WRAs* is of length zero. If this is the case, the *Aircraft* is immediately added to the list of *Aircraft* ready for a sortie in an event called "AddToReady." If the *Aircraft* is NMC, it is added to a list of

NMC *Aircraft* and cannot fly until all of its failed *WRAs* are removed and replaced. If it is determined that the *Aircraft* is NMC, the repair process begins by scheduling an event named "StartAircraftRepair".

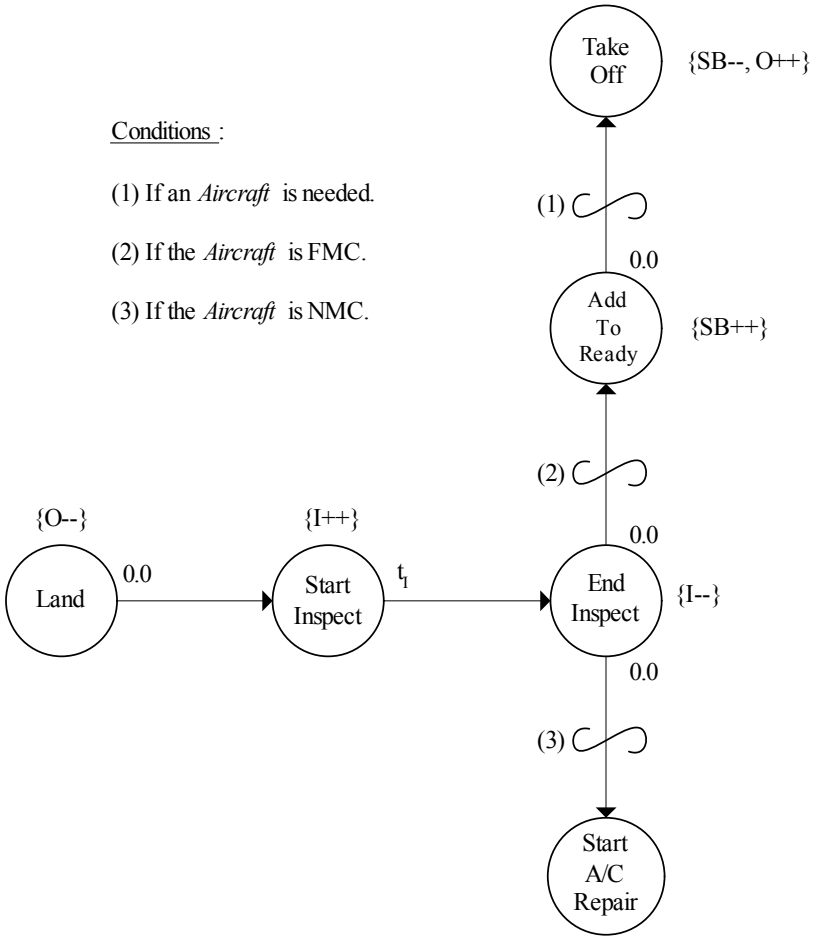


Figure 5 The Post-Flight Inspection Event Graph

In Figure 5, "O" and "SB" are defined as in the previous section. "I" is the number of *Aircraft* being inspected, and t_I is the InspectionTime. That is, the quantity of time the post-flight inspection takes. The "0.0" next to some of the arrow labels in Figure 5 means that the next event is scheduled without delay.

4. The Aircraft Repair Process

The StartAircraftRepair event starts the *Aircraft* repair process. All *WRAs* in the *Aircraft's* list of failed *WRA's* must be checked against the spares kit inventory. If a spare *WRA* matching that of a failed *WRA* is in stock in the spares kit, the spare *WRA* is removed from the inventory and a TTR generated from the MTTR random variate. This is done for each *WRA* on the list of failed *WRAs* and the generated TTRs are then summed. This sum is the amount of time necessary to repair the *Aircraft.* That is, the amount of time required for the repair process. The quantity of maintenance personnel (technicians) with the appropriate skill levels is assumed to be present. If a replacement *WRA* is not found to be in stock in the spares kit, the *Aircraft* is considered to be awaiting parts (AWP) and the failed *WRA* is added to the *Aircraft's* AWP list. An event named "EndAircraftRepair" is then scheduled as determined by the summed TTR.

If at the conclusion of event EndAircraftRepair the *Aircraft* is found to have *WRAs* that have not arrived from the requisition process, that is, the *Aircraft* has *WRAs* on its AWP list, the *Aircraft* remains idle until a *WRA* arrives that it needs. If a *WRA* arrives for this *Aircraft* while the *Aircraft* is in an active state of repair, the *WRA* is added to a list of additional *WRAs* that need to be installed at the conclusion of the current active repair session. If at the end of a repair session the *Aircraft* is found to have additional *WRAs* to install, TTRs for these additional *WRAs* are summed and another EndAircraftRepair event is scheduled per the newly summed TTR. This process repeats until the *Aircraft* is found to have all of its failed *WRAs* completely removed and replaced. When all of an *Aircraft's WRAs* are re-installed, the *Aircraft* is considered FMC, removed from the list of NMC *Aircraft,* and added to the list of *Aircraft* ready to fly a sortie. When a *WRA* is installed on an *Aircraft,* a new TTF is generated from the appropriate MTTF random variate.

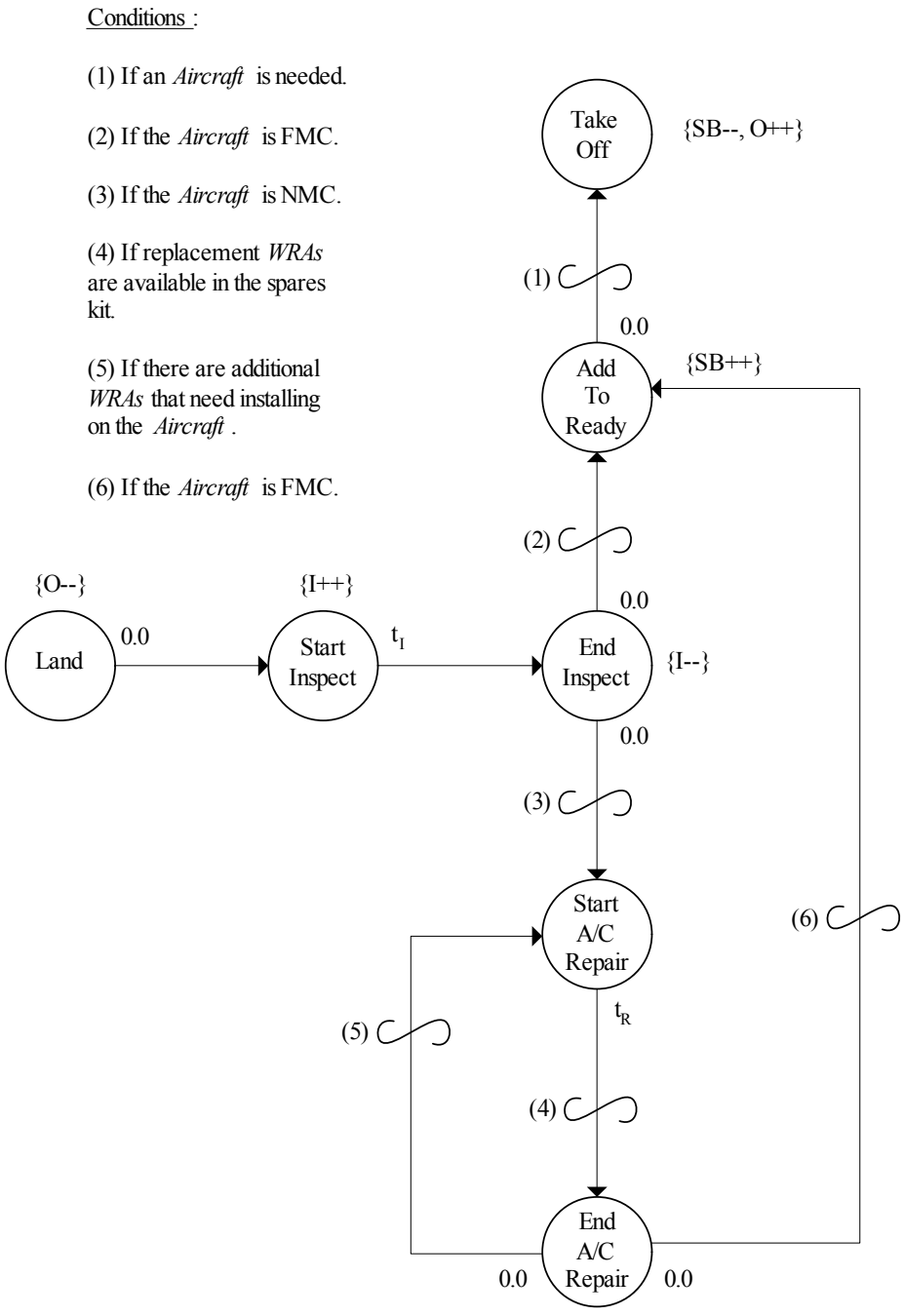


Figure 6 The Aircraft Repair Event Graph

In Figure 6, "SB, O, I," and t_I are defined as in previous sections and t_R is the RepairTime.

A *WRA* must be requisitioned to either replace the *WRA* removed from the spares kit or to replace the *WRA* not found to be in the spares kit inventory. SIMACE names the process of requisitioning a replacement *WRA* the "OST process".

5. The OST Process

WRAs are requisitioned to replace a *WRA* removed from the spares kit or to replace a *WRA* not found to be in the spares kit and therefore directly needed by an *Aircraft*. When a *WRA* is requisitioned, it sent to the detachment from an off-site location which causes a logistics delay time, called "ordering and shipping time" (OST).

When the need for a *WRA* exists, an event called "StartOST" is immediately scheduled. The OST random variate determined from the input data generates the amount of time to complete a requisition. The "EndOST" event is then scheduled for a duration determined (generated) by the OST random variate.

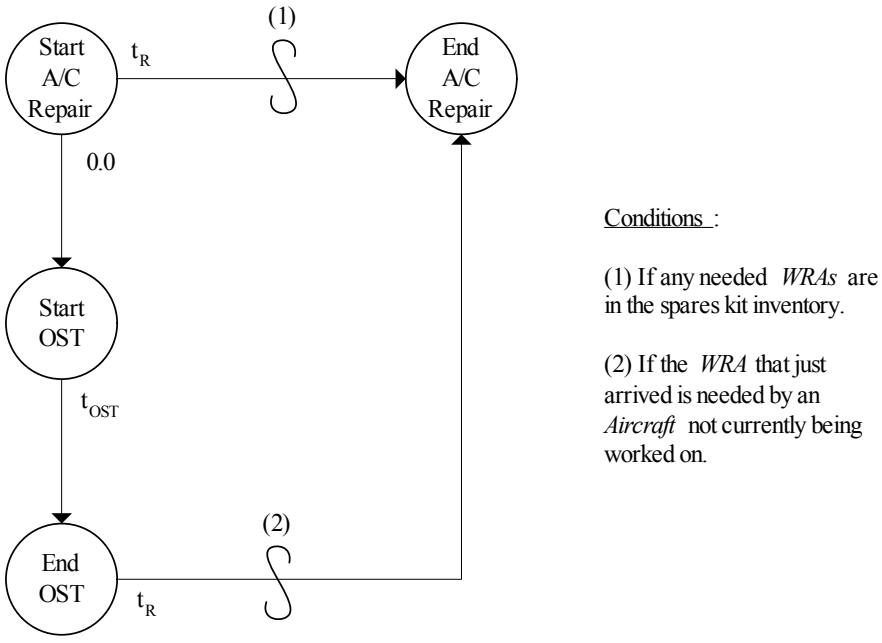


Figure 7 The OST Process Event Graph

In Figure 7, t_R is defined as in previous sections and t_{OST} represents the OST time for the *WRA*.

When a *WRA* arrives from the OST process, it is assigned to the *Aircraft* that will become FMC the quickest upon its receipt. The criterion used to determine the *Aircraft* most in need is by total number of AWP *WRAs*. That is, the newly arrived *WRA* is assigned to the *Aircraft* that has the least total number of AWP *WRAs*. If there is a tie for total number of AWP *WRAs,* the criterion selects the *Aircraft* with the most NMC time. If there is a tie between total number of AWP *WRAs* and the most NMC time, the criterion selects the *Aircraft* with the lowest tail number. An *Aircraft's* NMC time begins when the *Aircraft* is added to the list of NMC *Aircraft* (during event EndInspect) and ends upon its removal from the list of NMC *Aircraft* (during event EndAircraftRepair). If SIMACE determines that no *Aircraft* are in need of the newly arrived *WRA*, the *WRA* is assumed ordered as a replenishment item for the spares kit. In this case, the newly arrived *WRA* is added to the spares kit inventory.

G. EXPERIMENTAL DESIGN

The specific scenarios examined in this thesis assume a TransitTime value of 2-hours and a PatrolTime of 8-hours. Thus, the total time for each sortie is 12 flight hours.

Since the notional squadron is operating from a location distant to a supply warehouse, the scenarios assume OST to be Exponentially distributed with a mean of 169 hours (the number of hours in 1 week). While SIMACE allows for each *WRA* to have a different OST distribution, the scenarios assign all *WRAs* the same OST distribution of Exponential with mean 169. Of course, these values can be easily changed because of the robust design of SIMACE.

Using PC ARROWs, spares kit inventories were calculated, each forming a minimum cost spares kit inventory. Each of these spares kits (budget levels) was then run 100 times on SIMACE. The results were then plotted as to better visualize the cost and A_o relationship. The first portion of Chapter IV presents a brief comparison of PC ARROWs and SIMACE since output from PC ARROWs is used as input for SIMACE. Subsequent sections of Chapter IV present a few relationships (case study excursions)

involving a 90-day detachment of the notional aircraft. The first case study involves a detachment of four *Aircraft* with unmodified data (the Base Case). The second case study involves a detachment of five *Aircraft* with unmodified data (Case A). The third case study involves a detachment of four *Aircraft*, but whose data is modified (Case B). While Case B has four *Aircraft* on the detachment, the data is modified by improving the 50% lowest performing *WRA* failure times by 25% thereby increasing each affected *WRA* cost by 50%.

The scenarios assume each *Aircraft* added to the detachment incurs an additional cost of $50 million, which includes the cost of the additional *Aircraft* and the personnel/equipment that are necessary to support it. The cost of improved reliability affects the cost of each *Aircraft* as well. The additional cost per *Aircraft* incurred by improving the 50% lowest performing *WRA* failure times by 25% thereby increasing their cost by 50% is $4.77 million. This dollar amount was found by totaling the value of all *WRAs* installed on an *Aircraft*, with and without *WRAs* possessing improved reliability, and the difference calculated.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS AND ANALYSIS

A. INTRODUCTION TO RESULTS

The purpose of Chapter IV is to present results and analyses as introduced in Chapters I, II, and III and presents the data via figures. Please refer to Appendix C (Simulation Output) for the tables containing the data used to generate the figures in this chapter. For reviewing purposes, the spares kits are created in PC ARROWs and the analysis done using SIMACE.

This chapter first presents a brief comparison of PC ARROWs and SIMACE. The analysis continues with a comparison of the Base Case (4 *Aircraft,* unmodified data) against Case A (5 *Aircraft,* unmodified data), and Case B (4 *Aircraft,* improved *WRA* reliability) in order to determine if it is a better value to support an additional *Aircraft* at a cost of $50 million or pay more for increased *WRA* reliability with an additional cost of $4.77 million per *Aircraft*.

For each of the scenarios (case studies) presented in this chapter, minimum cost spares kit inventories (budget levels) to achieve A_o goals were calculated. A_o goals were started at 0.0 and incremented by 0.04 (4%) up to where A_o levels would not improve as a result of increasing the budget. Calculating spares kit inventories in this manner results in 162 spares kit inventories being calculated by PC ARROWs and evaluated using SIMACE. To facilitate the simulation of these data, each inventory listing is contained in its own unique XML document. Handling spares kit inventories in this manner facilitates the automation process of SIMACE.

B. PRESENTATION OF RESULTS

1. Relationship of SIMACE and PC ARROWs

Before the Base Case, Case A, and Case B are discussed, the relationship of SIMACE and PC ARROWs is presented. The first simulation runs were done to gain insight into the relative behavior(s) of SIMACE and PC ARROWs.

The figures that follow are labeled according to the following format: (TransitTime, PatrolTime, InspectionTime). For example, a label of "(2.0, 8.0, 2.0)" means that each *Aircraft* has a TransitTime to and from the patrol area of 2.0 hours, a PatrolTime of 8.0 hours, and a post-flight InspectionTime of 2.0 hours. Another example is "(0.0, 8.0, 0.0)" which is interpreted as a TransitTime to and from the patrol area of 0.0 hours, a PatrolTime of 8.0 hours, and a post-flight InspectionTime of 0.0 hours.

The relationship between SIMACE and PC ARROWs is presented via two cases: Case Y and Case Z. Case Y investigates a TransitTime, PatrolTime, and InspectionTime of (0.0, 8.0, 0.0), respectively. Case Z investigates a TransitTime, PatrolTime, and InspectionTime of (2.0, 8.0, 2.0), respectively.

Case Y and Case Z are run on SIMACE using two different methods for comparison with PC ARROWs:

Method 1) Assuming Exponentially distributed MTTF, constant MTTR, and constant OST (closest approximation to PC ARROWs).

Method 2) Using Exponentially distributed MTTF, Exponentially distributed MTTR, and Exponentially distributed OST.

Note the flight hour goal for Case Y is 2160 flight hours because of the TransitTime to and from the patrol area of 0.0 hours and the post-flight InspectionTime of 0.0 hours. That is, the TransitTime is not included in the calculation of the number of flight hours flown:

$$90 \text{ days} \times \frac{24 \text{ hours}}{\text{day}} = 2160 \text{ required flight hours} \qquad \text{(Equation 4.1)}$$

The flight hour goal for Case Z, as well as in the Base Case, Case A, and Case B, is 3240 flight hours as determined by the following equations:

$$90 \text{ days} \times \frac{24 \text{ hours}}{\text{day}} = 2160 \text{ required patrol hours} \qquad \text{(Equation 4.2)}$$

$$\frac{2160 \text{ required patrol hours}}{8 \text{ patrol hours per sortie}} = 270 \text{ sorties} \qquad \text{(Equation 4.3)}$$

$$270 \text{ sorties} \times 4 \text{ hours transit time per sortie} = 1080 \text{ transit hours} \qquad \text{(Equation 4.4)}$$

$$2160 \text{ patrol hours} + 1080 \text{ transit hours} = 3240 \text{ flight hours} \qquad \text{(Equation 4.5)}$$

Each simulation (design point) is determined by making 100 repetitions of SIMACE. An A_o is observed for each repetition and after the 100 repetitions, statistics were collected. Please refer to Appendix C for output data. On the figures that follow pertaining to Case Y and Case Z, information corresponding to Method (1) is labeled as "Constant" because of the constant MTTR and constant OST. Information corresponding to Method (2) is labeled as "Variable" because of the Exponentially distributed MTTR and Exponentially distributed OST. In both cases, the MTTF is treated as originating from an Exponential distribution.

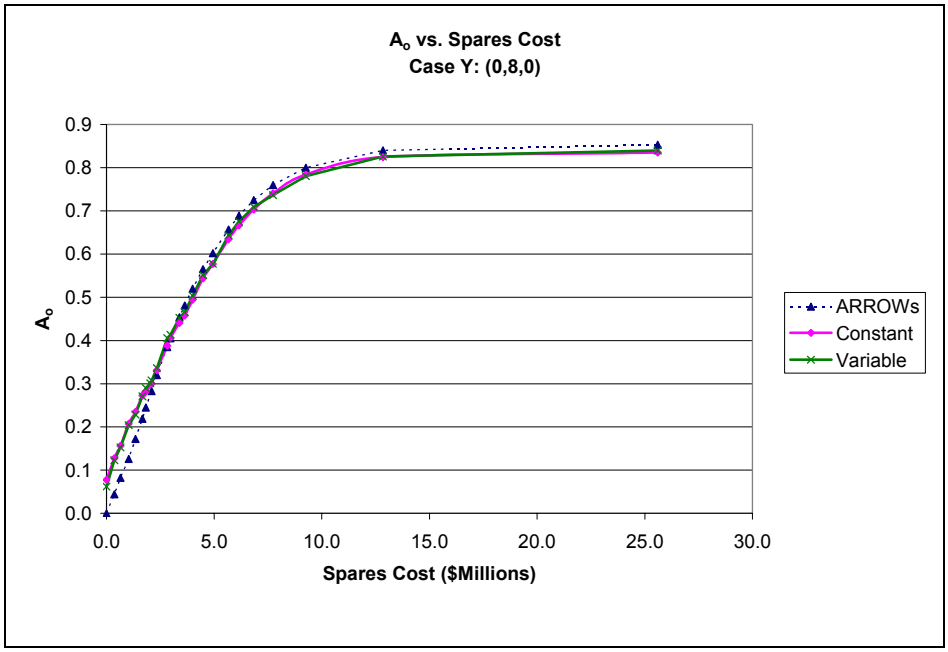


Figure 8 Case Y: (0, 8, 0)

Figure 8 shows that SIMACE produces results very similar to that of PC ARROWs. However, there is a slight disparity between SIMACE and PC ARROWs when the A_o goal is low. A possible reason for this is that SIMACE captures the fact that with zero *WRAs* in the spares kit inventory, there still remains a small degree of A_o in the system (inherent availability). When SIMACE begins a simulation run, all *Aircraft* are assumed to be operationally available, which is in keeping with how detachments operate in real, operational environments. That is, all *WRAs* are fully functioning on day one. It is only after flight operations begin that *WRAs* fail and must then be requisitioned. PC ARROWs, on the otherhand, uses steady-state methodology and does not capture inherent availability. Analysis suggests SIMACE produces a higher estimated A_o than PC ARROWs (statistically significant at level $\alpha = 0.05$) at lower spare kit inventories for the reasons described above.

SIMACE under Case Y (0, 8, 0) did not produce a statistically significant difference between using constant MTTR and constant OST vice variable MTTR and variable OST under the assumption of the variation originating from an Exponential distribution. However, using distributions other than the Exponential may produce different results as is demonstrated later in this chapter.

Figure 9 shows a difference between the A_o estimates from SIMACE and those from PC ARROWs near the end points under Case Z (2, 8, 2):

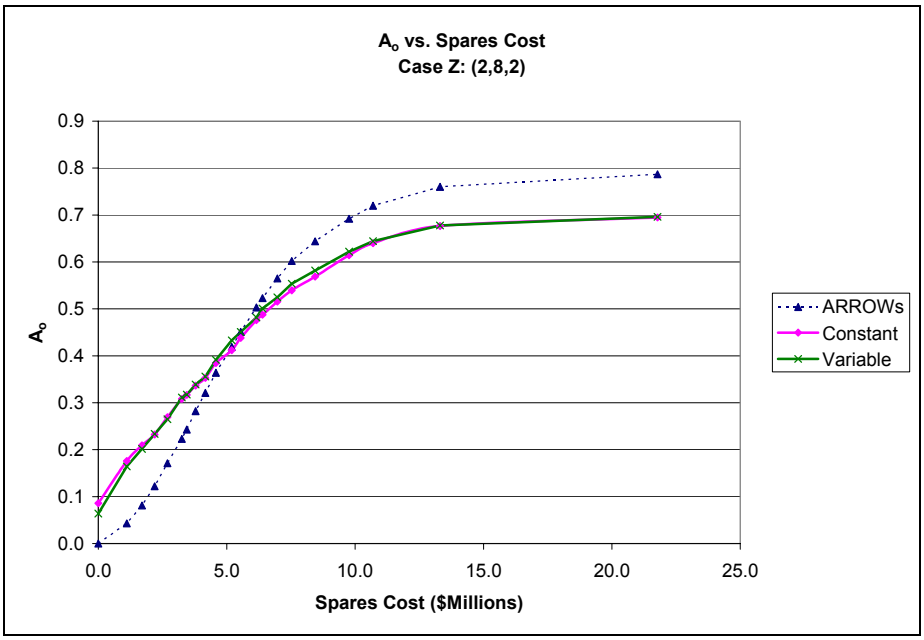


Figure 9 Case Z: (2, 8, 2), without InspectionTime

In Figure 9, a possible reason for the difference on the low end (as with Figure 8) is SIMACE capturing the fact that there exists inherent reliability. A possible reason for the difference on the high end may be due to differences in how A_o is computed in the two models. For example, SIMACE does not count InspectionTime as contributing to A_o whereas PC ARROWs does. Figure 10, below, shows how including InspectionTime as a contributor to A_o increases the estimates of SIMACE thereby causing a closer fit of the two models:

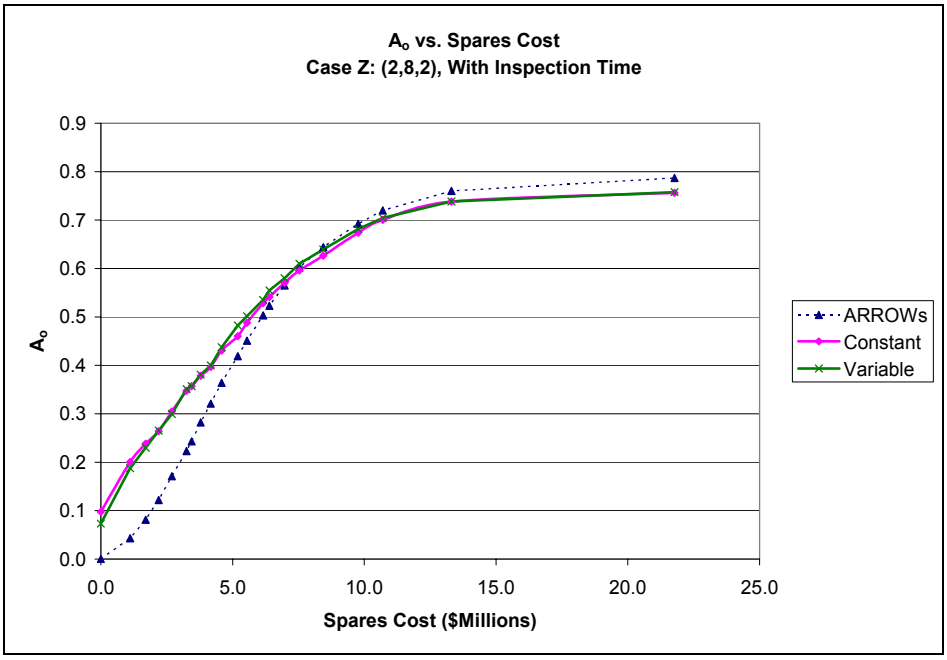


Figure 10 Case Z: (2, 8, 2), with InspectionTime

The comparison of SIMACE to PC ARROWs is now complete and it is concluded that SIMACE operates as intended. The remaining sections of Chapter IV utilizes SIMACE to evaluate investment trade-offs using A_o estimates as formulated in Case Z (without InspectionTime) (Figure 9). That is, comparisons between the Base Case (4 *Aircraft,* unmodified data) against Case A (5 *Aircraft,* unmodified data), and Case B (4 *Aircraft,* improved *WRA* reliability) are made based on the calculation of A_o as presented in Appendix B and modeled by SIMACE. Actual verification of the results with other models, to include PC ARROWs, is left for further research.

2. Base Case: 4 Aircraft, Unmodified Data

The Base Case (4 Aircraft, Unmodified Data) is used to compare against the excursions of Case A and Case B that follow.

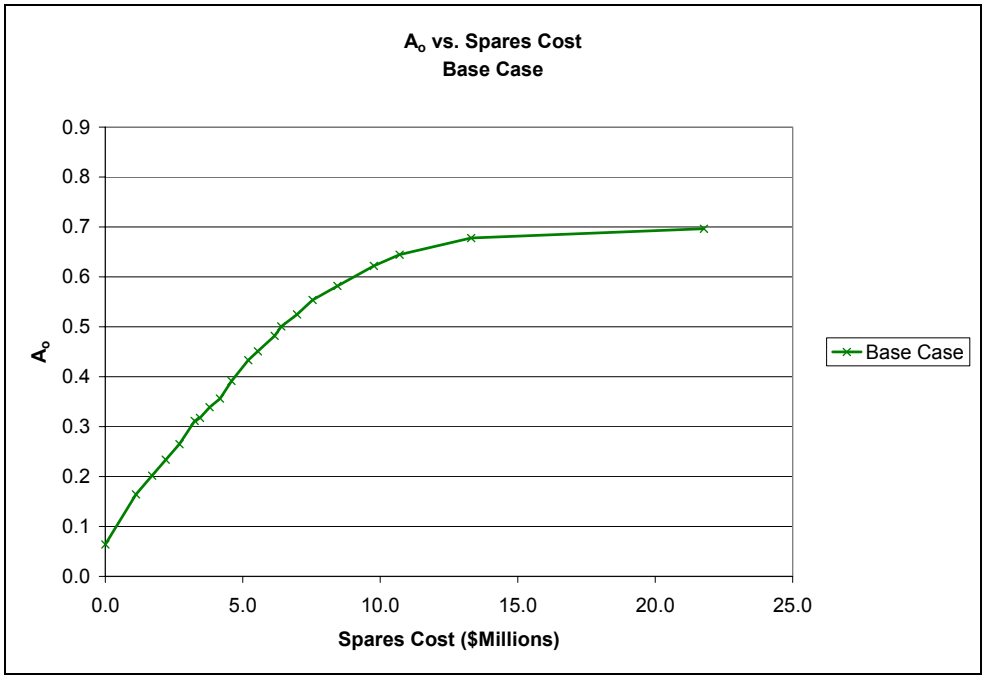


Figure 11 Base Case: (2, 8, 2), 4 Aircraft, Unmodified Data

Figure 11 is exactly the same data labeled as "Variable" in Case Z (Figure 9) and shows there is a limit to A_o (0.7064) even with an infinite spares budget. This value is found by running SIMACE with a spares kit inventory higher than would ever be needed for each *WRA* over the course of the 90-day detachment (quantity of 100 for each *WRA*). Likewise, the calculation of minimum A_o is found by running SIMACE with a spares kit inventory of zero (quantity of 0 for each *WRA*).

3. Case A: 5 Aircraft, Unmodified Data

In this excursion, Case A (5 Aircraft, Unmodified Data), the value of adding a fifth *Aircraft* is evaluated in terms of improved A_o. Figure 12, below, displays the relationship of the Base Case and Case A with respect to spares cost:

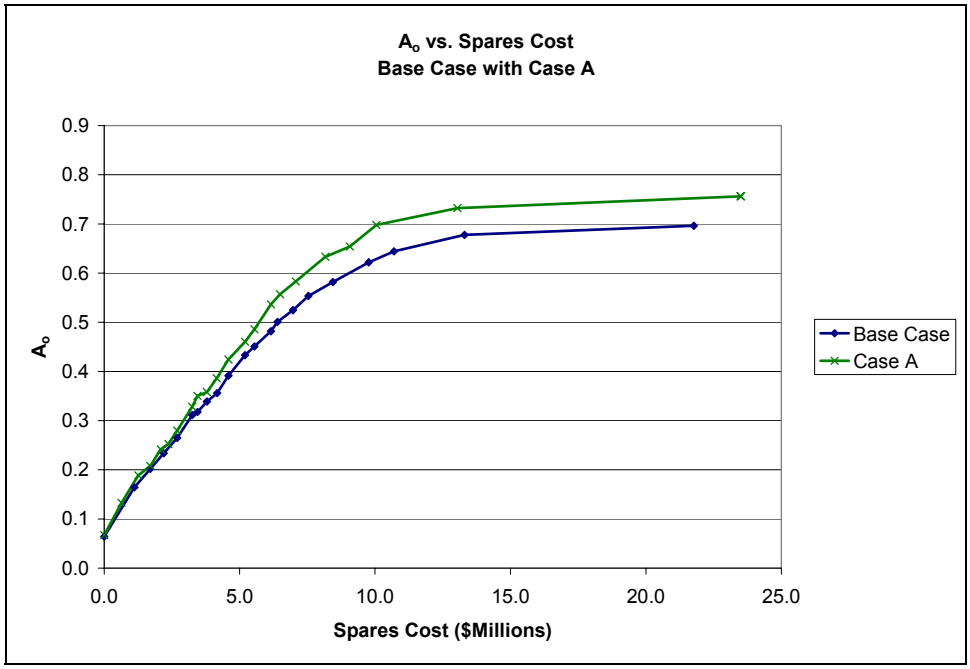


Figure 12 Base Case with Case A (5 Aircraft, Unmodified Data)

A major portion of the projected life-cycle cost for a given system results from the consequences of decisions made during early planning and as part of system conceptual design (Fabrycky, 1998). Figure 13, below, displays the relationship of the Base Case and Case A with respect to total cost (spares cost and *Aircraft* acquisition cost):

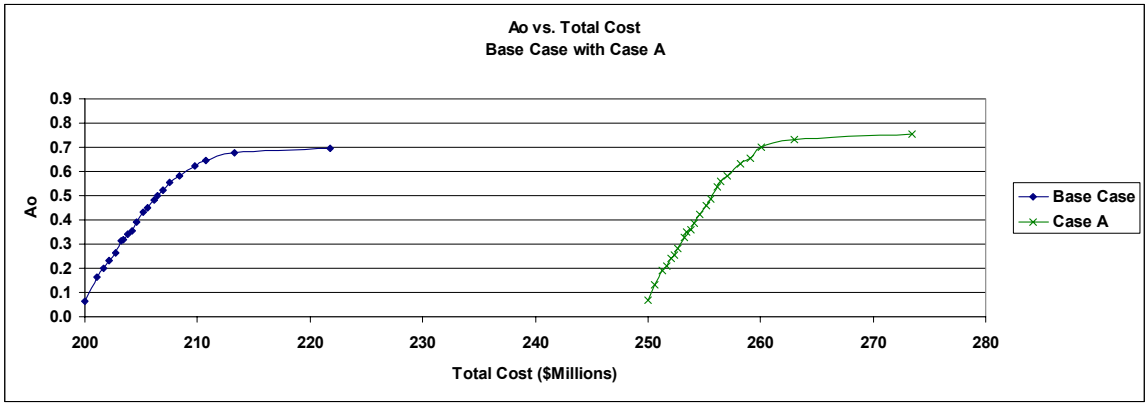


Figure 13 Base Case with Case A, A_o vs. Total Cost

Figure 13 may initially appear it is not cost-effective to purchase the 5th *Aircraft*. However, in addressing this economic issue one must look at total cost in the context of the overall life cycle. For example, note in Figure 12 that to achieve an A_o of 0.6963 in the Base Case, it is necessary to pack a spares inventory valued at $21.77 million. Note also that in order to achieve an A_o of 0.6981 in Case A, it is necessary to pack a spares inventory valued at $10.05 million. This means that additional funds, called recurring costs, of as much as $11.72 million must be budgeted above that of Case A (5 *Aircraft*) when choosing the Base Case (4 *Aircraft*) every time there is a 90-day detachment. This assumes that all spares in the prior detachment's spares inventory were used, which is unlikely.

Not only is the spares budget lower for Case A with respect to the Base Case for approximately the same level of A_o (in this example approximately 0.70), but the spares inventory "footprint" is lower for Case A than for the Base Case. Under the Base Case (with A_o = 0.6963), there are 615 *WRAs* in the spares inventory and under Case A

(with A_o = 0.6981), there are 369 *WRAs* in the spares inventory. There are less *WRAs* necessary in Case A's spares inventory because fewer flight hours are being accumulated per *WRA* compared to the Base Case. Fewer flight hours per *WRA* suggests the quantity of maintenance personnel can be reduced in Case A thereby reducing personnel cost. Reducing the quantity of flight hours per *WRA* also reduces the quantity of supply requisitions thereby reducing logistics delay time and the cost to transport replacement *WRAs* from various points of origin to the detachment site. This would also reduce the quantity of *Aircraft* providing logistical support to the detachment thereby increasing the quantity of *Aircraft* available for operational missions. Case A requires a greater up-front investment (an additional $50 million) than the Base Case, thus this excursion suggests it may be of greater value to invest in the additional *Aircraft* because of the reduced recurring costs as described above.

While in this thesis SIMACE is being used to model a limited deployment of 90-days for 4 and 5 *Aircraft*, it may be possible for SIMACE to aid in total life cycle cost analysis with additional modifications made possible by its extensible design. Examples of how SIMACE may be extended include adding a module which monitors the quantity of each type *WRA* that is removed and replaced for the time period specified by the user (SIMACE input parameter = DeploymentLength) thus aiding in the analysis of the quantity of maintenance personnel necessary to man each work center (personnel cost). A more detailed analysis of personnel costs could be conducted by adding another module to SIMACE modeling the actual activity of personnel within each work center (queues of maintenance workers). SIMACE may also be run for several scenarios to model activities at different bases of operation. Using SIMACE in this manner could assist in determining the quantity of *Aircraft* necessary to meet ongoing fleet wide operational and training requirements ("fleet sizing").

4. Case B: 4 Aircraft, Improved Reliability

In this excursion, Case B (4 Aircraft, Improved Reliability), the value of improved reliability in terms of improved A_o is investigated. Figure 14, below, displays the relationships of the Base Case, Case A, and Case B with respect to spares cost:

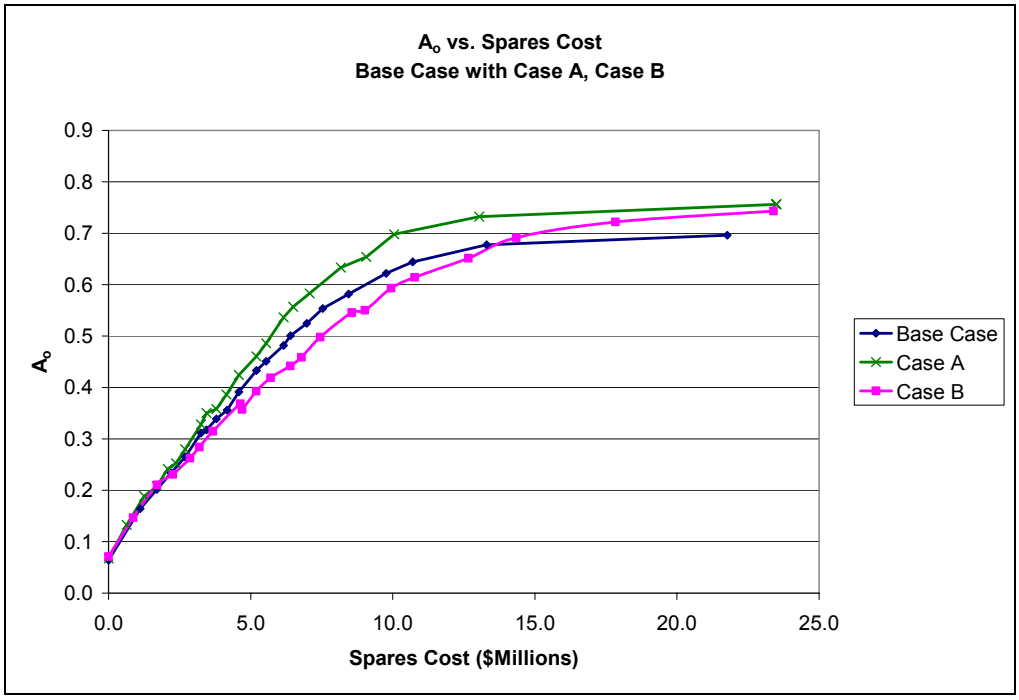


Figure 14 Base Case with Case A, Case B (4 Aircraft, Improved Reliability)

Figure 15, below, displays the relationships of the Base Case, Case A, and Case B with respect to total cost (spares cost and *Aircraft* acquisition cost):

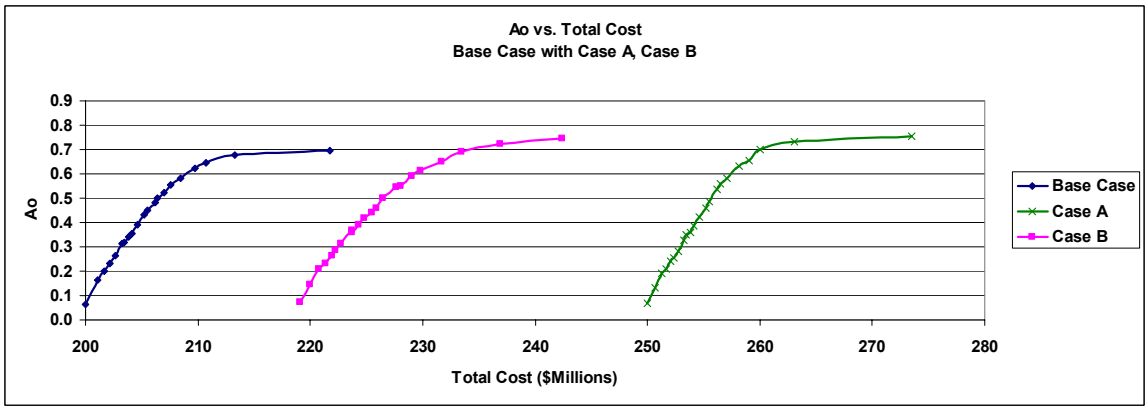


Figure 15 Base Case with Case A and Case B, A_o vs. Total Cost

Note that in comparison to the Base Case, Case B requires an additional upfront investment of $4.77 million per *Aircraft* due to the cost associated with increased reliability:

$$\$4.77 \text{ million} \times 4 \textit{ Aircraft} = \$19.08 \text{ million} \qquad \text{(Equation 4.6)}$$

Thus, the additional investment (baseline) for Case B is $19.06 million more than the Base Case scenario.

Figure 15 may initially appear to show it is less value to pay 50% more for a MTTF improvement of 25% for the 50% least reliable *WRAs* as in Case B than it is to follow the assumptions of the Base Case (4 *Aircraft,* Unmodified Data). However, as in the relationship of the Base Case to Case A, this may not be entirely true. For example, note in Figure 14 that to achieve an A_o of 0.6963 in the Base Case it is necessary to pack a spares inventory valued at $21.77 million. Also note that in order to achieve an A_o of 0.6915 in Case B it is necessary to pack a spares inventory valued at $14.35 million. To achieve approximately the same level of A_o , additional funds of $7.42 million must be budgeted above that of Case B (4 *Aircraft,* Improved Reliability) with respect to spares cost when choosing the Base Case (4 *Aircraft,* Unmodified Data). As described in the previous section, this may have implications if total life cycle costs are considered vice that of a 90-day detachment.

Not only is the spares cost lower for Case B with respect to the Base Case for an A_o of approximately 0.70, but the spares inventory "footprint" is lower for Case B than for the Base Case. For example, under the Base Case (with A_o = 0.6963), there are 615 *WRAs* in the spares inventory and under Case B (with A_o = 0.6915) there are 349 *WRAs* in the spares inventory. Improving reliability, as in Case B, lowers the quantity of spare parts necessary to achieve approximately the same level of A_o in the Base Case. Not only would reducing the spares inventory as a result of improved reliability lower the logistics requirement for transporting it from place to place, it would also lower the quantity of

requisitions over the course of the deployment period. In addition, improving reliability may reduce the quantity of personnel necessary to man each work center thus reducing personnel costs. As described in the previous excursion, future modifications to SIMACE will enable it to conduct a more detailed analysis of personnel costs.

Note that while the quantity of *WRAs* in the spares inventory is reduced in Case B, the average cost per *WRA* is actually more than that of the Base Case. For example, under the Base Case (with A_o = 0.6963), there are 615 *WRAs* valued at $21.77 million in the spares inventory and under Case B (with A_o = 0.6915) there are 349 *WRAs* valued at $14.35 million in the spares inventory. For approximately the same level of A_o, the average cost per *WRA* for the Base Case is $0.0354 million and the average cost per *WRA* for Case B is $0.0411 million. The average cost per *WRA* is more for Case B than for the Base Case, but note that Case B requires a lower budget for spares due to the cost associated with increasing reliability.

Under Case A (A_o = 0.6981), there are 369 *WRAs* valued at $10.05 million in the spares inventory. This implies that the average cost per *WRA* for Case A is $0.0272 million. If an A_o goal of approximately 0.70 is desired, it is necessary to budget an additional $4.30 million (recurring) for spares when choosing the scenario of Case B over Case A.

When considering only a single segment of life-cycle cost (such as budget for spares), one must be sure that decisions are not based on that one segment alone without the consideration of the overall effects on total life-cycle cost (Fabrycky, 1998). Life-cycle costing includes a variety of factors reflecting different types of activities. SIMACE is a simulation model that can immediately assist in the analysis of several of these factors and can be expanded in the future by adding modules as the need for more detailed analyses grows.

5. Application of the Weibull Distribution to MTTF

The purpose of this section is to demonstrate the flexability of SIMACE by replacing the Exponential distributions with Weibull distributions for the MTTF in the Base Case.

As previously stated in Chapter III (Simulation Model Development), reliability data such as MTTF are frequently modeled using a Weibull distribution. However, the scenarios presented in this thesis have thus far only used the Exponential random variate to generate TTFs due to the lack of data to parameterize other distributions.

The two-parameter Weibull distribution is defined by a shape parameter called "alpha" (α) and a scale parameter called "beta" (β) which are both defined on $[0,\infty)$. The Weibull distribution is the same as the Exponential distribution when $\alpha = 1$. That is, β is simply the mean of the Exponential distribution when $\alpha = 1$. Shape parameters (α) greater than one are used to illustrate "wear-out" characteristics of components. To approximate Weibull distributions from the pre-existing MTTF point estimates, this thesis uses a value of $\alpha = 1.2$ for all MTTF Weibull random variate approximations. A value of $\alpha = 1.2$ is chosen because previous research shows this value is adequate to represent "wear-out" chacteristics (Werenskjold, 1998). The investigation into the effects of stronger degrees of "wear-out", thus higher values of the shape parameter α, is left for further research.

Weibull random variate means are given by the following equation (Law, 2000):

$$\mu = \frac{\beta}{\alpha}\Gamma\left(\frac{1}{\alpha}\right) \qquad \text{(Equation 4.7)}$$

Since in this study α is always assigned the value of 1.2, (Equation 4.7) becomes:

$$\mu = \frac{\beta}{1.2}\,\Gamma\left(\frac{1}{1.2}\right) = \frac{\beta}{1.2} \times 1.1288 = \beta \times 0.9407 \qquad \text{(Equation 4.8)}$$

Therefore, a β is calculated for each MTTF according to (Equation 4.9):

$$\beta = \frac{\mu}{0.9407} \qquad \text{(Equation 4.9)}$$

where μ is the mean of the MTTF Exponential random variate being converted.

Figure 16 is a dual plot of the Base Case showing the effect of using the Weibull random variates vice the Exponential random variates in the generation of TTFs:

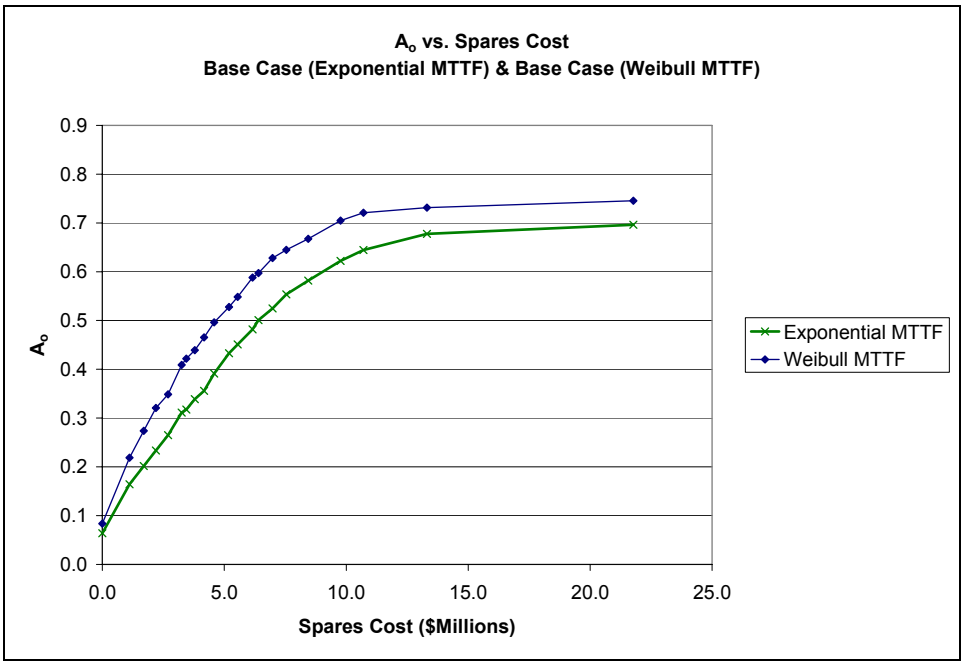


Figure 16 Base Case (Exponential MTTF) and Base Case (Weibull MTTF)

Analysis of Figure 16 reveals there is a statistically significant difference (at level $\alpha = 0.05$) between all A_o estimates generated using the Exponential distribution and those generated using the Weibull distribution. That is, in this case the Exponential distribution is more conservative than the Weibull distribution. A_o as a function of Total Cost (*Aircraft* and spares costs) is not examined because both curves in Figure 16 originate from the Base Case (4 *Aircraft,* Unmodified Data). A copy of the simulation output used to generate Figure 16 is included in Appendix C (Simulation Output).

This example is a presentation of the significantly different results that may be obtained by using distributions other than the Exponential. Thus the assumption about the distribution of the MTTF random variate is critical and is a strong reason why data collection systems should provide more information than just the mean. The testing of additional distributions is left for further research.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The purpose of this thesis was to create an upgradeable simulation tool that can be used to investigate relationships that affect cost and operational availability of the P-3 replacement aircraft (notional) on a deployment and exercise the simulation tool to evaluate trade-offs in achieving various levels of A_o. The purpose of exercising the tool was to show its flexability and value as a "proof of concept" for future analyses. The specific results should be seen as representational of the kinds of analysis that can be done with SIMACE.

The analysis presented in Chapter IV suggests that under the assumption of cases investigated in this thesis, it may be of greater long-term value to invest funds in an additional *Aircraft* than to invest in increased reliability (improving the 50% lowest performing *WRAs* by 25% at an increased cost of 50%). However, there is more that goes into estimating life cycle costs than just the consideration of the spares inventory. Emphasis should rest not just on an individual cost element such as spares cost, but rather on total life cycle cost. SIMACE can be expanded to assist in a more detailed analysis of factors that affect total life cycle cost for not only the replacement *Aircraft* for the P-3, but for any *Aircraft* a user has appropriate data for.

PC ARROWs creates minimum cost spares kits based upon the assumption that the requisition system, and *WRA* failures, behave according to a Poisson process. In reality, requisition systems and failure times may not always follow this assumption.

This thesis presented one scenario using Weibull random variates instead of Exponential random variates to generate TTFs. The results detected a statistically significant difference (at level $\alpha = 0.05$) between what the two distributions produced in terms of A_o estimates. This shows that using distributions other than the Exponential are capable of producint significantly different results.

More replications of SIMACE simulation runs do not produce statistically significant results at level $\alpha = 0.05$. This comparison was made using results obtained

from 100 simulation runs and 500 simulation runs for each design point. Due to the amount of time each simulation run takes, the testing of higher simulation repetitions is left for additional research.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

There are a number of items that SIMACE does not take into consideration due to the time constraints of completing this thesis. However, SIMACE is developed in such a way that it can be easily modified to incorporate additional properties that could possibly do a better job of modeling an authentic operating environment. Below are some recommendations to enhance the "resolution" of SIMACE:

Incorporate both a priority OST and a routine OST.

Use more realistic random variate distributions as discussed in Chapter III.

It may be desirable to schedule the removal and replacement of some *WRAs* during the detachment. For example, an engine may be scheduled for removal and replacement after 2000 flight hours in order to minimize the probability of a failure during flight. This is a desired property if SIMACE is developed into more of a "steady-state" model.

An I-Level maintenance process should be incorporated into the model.

SIMACE may possibly assist in the efforts to determine a statistical model for forecasting aircraft utilization rates to assist in the improvement of the "productive ratio" of Naval aviation's aircraft inventory.

Another method to approximate the relationship between budget for spares, OST, MTTF, A_o, etc. may be to utilize one or more of the robust simulation techniques developed over the past few years: Professor Jack Kleijnen of Tilburg University (The Netherlands) researches robust solutions aimed at finding appropriate values for the factors that decision makers can control, while accounting for the randomness of the uncontrollable environmental factors. He uses Latin Hypercube sampling, estimates the controllable factor values that minimize the output's expected value and variance, derives

a confidence region, and selects a robust solution. Professor Bruce Schmeiser of Purdue University estimates a function at any specified point by using stochastic root finding via retrospective approximation. The possibility exists that SIMACE could be transformed into a stand-alone RBS tool by applying one of the above, or possibly other, advanced simulation techniques.

Developing a simulation model is a very difficult and time-consuming endeavor even for the most "trivial" of logistical processes. Perhaps Dr. Sherbrooke, formerly of the Logistics Management Institute, summarizes this difficulty the best:

> In logistics applications, we know that all of our data are estimates: demand rates, costs, lead times, repair times, Pr{local repair capability}, etc. We know that we will never hit the projected availability or cost precisely in the real world, regardless of the degree of mathematical sophistication employed. (Sherbrooke, 2000)

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. LIST OF ACRONYMS

A_o	Operational Availability
ARROWS	Aviation Retail Requirements Oriented to Weapon Replaceable Assemblies
AWP	Awaiting Parts
FMC	Fully Mission Capable
LMDSS	Logistics Management Decision Support System
MADT	Mean Aircraft Down Time
MC	Mission Capable
MDT	Mean Down Time
MLDT	Mean Logistics Delay Time
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
NALDA	Naval Aviation Logistics Data Analysis
OT	Operational Time
NAVAIR	Naval Air Systems Command
NAVICP	Naval Inventory Control Point
NIIN	National Item Identification Number
NMC	Not-Mission Capable
PC ARROWS	Personal Computer Aviation Retail Requirements Oriented to Weapon Replaceable Assemblies
RBS	Readiness Based Sparing
RNG	Random Number Generator
ST	Stand-By Time
TTF	Time to Failure
TTR	Time to Repair
WRA	Weapons Replaceable Assembly
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. CALCULATION OF OPERATIONAL AVAILABILITY

The purpose of Appendix B is to give the reader a better understanding of A_o by providing an overview of some of the various methods used to calculate A_o .

A_o as used in this thesis represents the expected percentage of time that a weapon system will be ready to perform satisfactorily in an operating environment when called for at any random point in time (OPNAVINST 3000.12, 1987).

A_o can be expressed most fundamentally by the following equation:

$$A_o = \frac{\text{uptime}}{\text{uptime} + \text{downtime}} = \frac{\text{uptime}}{\text{total time}} \qquad \text{(Equation B.1)}$$

Uptime is the time during which the system is in condition to perform its required functions. Downtime is the time during which the system is not in condition to perform its required functions. The "system" in question throughout this thesis is the P-3 replacement aircraft (notional) and if it is not capable of functioning it is considered not-mission capable (NMC).

Although (Equation B.1) may provide a reasonably accurate estimate of A_o it does not provide the level of detail necessary to determine what specific factors affect it. To determine causes of poor A_o , the effects of controllable factors on uptime and downtime must be determined with respect to reliability, maintainability, and supportability. Each of these is defined as follows:

> Reliability is the probability that an item can perform its intended function for a specified interval under stated conditions. It is controllable primarily by design and secondarily by ensuring that a system is used in the manner for which it was designed.
>
> Maintainability is the measure of the ability of an item to be retained in or restored to specified condition when maintenance is performed by

personnel having specified skill levels, using prescribed procedures and resources, at each prescribed level of maintenance and repair.

Supportability is the ability to satisfy material and administrative requirements associated with restoring the operation of failed system or equipment (OPNAVINST 3000.12, 1987).

(Equation B.1) can now be broken down to the following:

$$A_o = \frac{\text{MTBF}}{\text{MTBF} + \text{MDT}} \qquad \text{(Equation B.2)}$$

where MTBF = mean time between a system failure and MDT = mean down time of the system. MTBF is defined as the total time that the system is in an "up" status divided by the number of failures during that time period. MDT is defined as the total time the system is in a "down" status divided by the number of failures during that time period. MTBF is interpreted as the *uptime* and MTBF + MDT is interpreted as the *total time*.

The maintenance component of A_o is the average amount of time required to repair a failed WRA at the organizational level (O-level) of maintenance when all resources are available. Call this the mean time to repair (MTTR). The supply (logistical delay) component is the average amount of delay caused by the logistical support system. Let this be the mean logistical delay time (MLDT).

With respect to reliability, maintainability, and supportability, MTBF is a measure of reliability, MTTR a measure of maintainability, and MLDT a measure of supportability. Now (Equation B.2) can be modified to following:

$$A_o = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR} + \text{MLDT}} \qquad \text{(Equation B.3)}$$

Systems can be described as falling into one of three categories: continuous-use, intermittent use, or impulse. Continuous-use systems are systems that are always in use. For example, cell phone towers and an emergency exit sign inside an office building. Intermittent-use systems are systems that have long periods of stand-by time between operational uses. For example, a car, computer printer, or aircraft. Impulse systems are systems that are usually used once or for an extremely short period of time between uses such as the starter on an automobile.

For continuous-use systems, mean-calendar time between failures (MCTBF) is the same as mean operating time between failures (MOTBF) because the system is always in use. In addition, the use of MTBF is consistent with the idea of measuring uptime in terms of calendar time. This is convenient since all downtime is measured in terms of calendar time. For continuous-use systems, (Equation B.3) should be used for the calculation of A_o .

For intermittent-use systems such as car headlights, MOTBF is not the same as MCTBF, so the MTBF must be weighted. For aircraft systems, A_o is commonly defined as:

$$A_o = 1 - \frac{\text{MTTR} + \text{MLDT}}{\text{K'(MTBF)}} \qquad \text{(Equation B.4)}$$

where

$$\text{MTBF} = \frac{\text{UPTIME}}{\text{\#FAILURES}} \qquad \text{(Equation B.5)}$$

and

$$K' = \frac{\text{TOTAL CALANDAR TIME}}{\text{TOTAL OPERATING TIME}} \qquad \text{(Equation B.6)}$$

Note: K' is the inverse of the utilization rate.

(Equation B.1) through (Equation B.6) are used for systems where the MTTF, MTTR, and MLDT are known quantities of the system. While this thesis could use (Equation B.4) through (Equation B.6) for the calculation of A_o, the model developed in this thesis collects "real time" data via a simulation vice estimating the individual parameters of system MTTF, system MTTR, and system MLDT. This thesis uses (Equation B.7) to calculate A_o:

$$A_o = \frac{\text{OT + ST}}{\text{TOTAL TIME}} \qquad \text{(Equation B.7)}$$

where *TOTAL TIME* (the denominator) equals the sum of operational time (OT), stand-by time (ST), total corrective maintenance time, total inspection time, and total logistics delay time.

Impulse-use systems calculate A_o as follows:

$$A_o = \frac{\text{TOTAL NUMBER OF SUCCESSES}}{\text{TOTAL NUMBER OF ATTEMPTS}} \qquad \text{(Equation B.8)}$$

It is important to note the difference between the aircraft status of "FMC" and that of "operational available." An aircraft considered to be FMC does not necessarily mean the aircraft is operationally available. When tracking aircraft status, fleet squadrons commonly assign the status of an aircraft in the morning and for reporting purposes the aircraft has this status for the next 24-hour period regardless of the aircraft's operational availability. For example, an aircraft is assigned as FMC at 0700 13 February 2003 holds

this status until the next morning at 0700 because the following day is a weekday. If the reporting period begins at 0700 on a Friday, the aircraft will hold its status until Monday morning at 0700. This thesis assumes 24-hour coverage in a wartime scenario and will collect data not based upon an aircraft's status every morning at 0700, but will continuously observe and collect the data required by (Equation B.7) to calculate A_o.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SIMULATION OUTPUT

The purpose of Appendix C is to display the data used to generate the charts throughout this thesis. The first column in each table (Goal A_o) represents the A_o goal provided to PC ARROWs in order to calculate the minimum cost spare kits. The second column (Spares Budget) is what PC ARROWs determines to be the cost of the spares kit to meet the A_o goal. The third column is simply the Spares Budget in units of millions of dollars. The fourth column (ARROWs A_o) is the A_o obtained by PC ARROWs. Please note that this value may not be exactly the same as the "Goal A_o." The fifth column (SIMACE mu A_o) represents the mean A_o obtained from 100 repetitions of SIMACE for that particular spares kit. Note that the calculation of A_o in this column does not take into consideration InspectionTime. The sixth column (SIMACE sigma A_o) represents the standard deviation of the mean A_o in the previous column. The seventh column (SIMACE mu FMC) represents the mean A_o obtained from 100 repetitions of SIMACE including InspectionTime in the calculation of A_o. The eighth column (FMC sigma A_o) represents the standard deviation of the mean A_o in the previous column.

The last two columns of data in each table are provided to the reader but were not commented on in this thesis. These columns pertain to the calculation of a FMC rate as follows. It could be the case that during a sortie a *WRA* fails thereby making the *Aircraft* not FMC, but possibly MC. The last two columns take this into consideration meaning that the time not FMC and flying is not included in the calculation of A_o. These columns are labeled as "SIMACE mu otFMC" and "SIMACE sigma otFMC" which pertain to the mean and standard deviation of the FMC rate obtained from 100 repetitions of SIMACE.

4 Aircraft
0.0 hr TransitTime, 8.0 hr PatrolTime, 0.0 hr InspectTime
Variable MTTF, Constant MTTR, Constant OST

reps = 100

Goal Ao	Spares Budget	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0775	0.0086	0.0775	0.0086	0.0510	0.0086
0.04	365632	0.37	0.0440	0.1276	0.0177	0.1276	0.0177	0.0847	0.0163
0.08	648441	0.65	0.0820	0.1562	0.0222	0.1562	0.0222	0.1061	0.0200
0.12	1033257	1.03	0.1260	0.2071	0.0286	0.2071	0.0286	0.1465	0.0267
0.16	1349097	1.35	0.1720	0.2344	0.0337	0.2344	0.0337	0.1677	0.0312
0.20	1670262	1.67	0.2190	0.2728	0.0349	0.2728	0.0349	0.1990	0.0329
0.24	1832271	1.83	0.2450	0.2815	0.0418	0.2815	0.0418	0.2056	0.0384
0.28	2094848	2.09	0.2830	0.3001	0.0411	0.3001	0.0411	0.2213	0.0382
0.32	2341690	2.34	0.3200	0.3313	0.0443	0.3313	0.0443	0.2488	0.0412
0.36	2826879	2.83	0.3850	0.3881	0.0489	0.3881	0.0489	0.2967	0.0464
0.40	2968401	2.97	0.4050	0.4063	0.0503	0.4063	0.0503	0.3132	0.0480
0.44	3390074	3.39	0.4540	0.4406	0.0573	0.4406	0.0573	0.3434	0.0548
0.48	3636777	3.64	0.4810	0.4577	0.0561	0.4577	0.0561	0.3590	0.0532
0.52	4001668	4.00	0.5200	0.4943	0.0599	0.4943	0.0599	0.3926	0.0575
0.56	4487259	4.49	0.5650	0.5440	0.0553	0.5440	0.0553	0.4395	0.0528
0.60	4941794	4.94	0.6020	0.5773	0.0576	0.5773	0.0576	0.4701	0.0561
0.64	5668848	5.67	0.6570	0.6343	0.0581	0.6343	0.0581	0.5241	0.0586
0.68	6148511	6.15	0.6890	0.6665	0.0456	0.6665	0.0456	0.5559	0.0465
0.72	6840474	6.84	0.7250	0.7031	0.0487	0.7031	0.0487	0.5918	0.0486
0.76	7739007	7.74	0.7600	0.7404	0.0421	0.7404	0.0421	0.6283	0.0426
0.80	9257914	9.26	0.8000	0.7841	0.0346	0.7841	0.0346	0.6716	0.0354
0.84	12835873	12.84	0.8400	0.8246	0.0183	0.8246	0.0183	0.7106	0.0205
0.88	25599156	25.60	0.8530	0.8351	0.0161	0.8351	0.0161	0.7218	0.0193
Approximate Max Ao >>	> 25599156	> 25.60	0.8530	0.8421	0.0080	0.8421	0.0080	0.7280	0.0132

Table 1 Case Y (Constant) Output

4 Aircraft
0.0 hr TransitTime, 8.0 hr PatrolTime, 0.0 hr InspectTime
Variable MTTF, Variable MTTR, Variable OST

reps = 100

Goal Ao	Spares Budget	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0619	0.0091	0.0619	0.0091	0.0396	0.0076
0.04	365632	0.37	0.0440	0.1230	0.0231	0.1230	0.0231	0.0822	0.0189
0.08	648441	0.65	0.0820	0.1528	0.0284	0.1528	0.0284	0.1036	0.0237
0.12	1033257	1.03	0.1260	0.2040	0.0346	0.2040	0.0346	0.1433	0.0304
0.16	1349097	1.35	0.1720	0.2286	0.0463	0.2286	0.0463	0.1624	0.0385
0.20	1670262	1.67	0.2190	0.2710	0.0467	0.2710	0.0467	0.1976	0.0415
0.24	1832271	1.83	0.2450	0.2901	0.0636	0.2901	0.0636	0.2141	0.0556
0.26	2037066	2.04	0.2750	0.3009	0.0579	0.3009	0.0579	0.2230	0.0508
0.28	2094848	2.09	0.2830	0.3075	0.0575	0.3075	0.0575	0.2273	0.0502
0.32	2341690	2.34	0.3200	0.3360	0.0578	0.3360	0.0578	0.2537	0.0528
0.36	2826879	2.83	0.3850	0.4055	0.0681	0.4055	0.0681	0.3143	0.0573
0.40	2968401	2.97	0.4050	0.4131	0.0729	0.4131	0.0729	0.3198	0.0673
0.44	3390074	3.39	0.4540	0.4524	0.0797	0.4524	0.0797	0.3547	0.0754
0.48	3636777	3.64	0.4810	0.4657	0.0678	0.4657	0.0678	0.3665	0.0639
0.52	4001668	4.00	0.5200	0.5037	0.0856	0.5037	0.0856	0.4042	0.0808
0.56	4487259	4.49	0.5650	0.5516	0.0718	0.5516	0.0718	0.4477	0.0687
0.60	4941794	4.94	0.6020	0.5777	0.0769	0.5777	0.0769	0.4716	0.0731
0.64	5668848	5.67	0.6570	0.6437	0.0672	0.6437	0.0672	0.5336	0.0663
0.68	6148511	6.15	0.6890	0.6758	0.0742	0.6758	0.0742	0.5677	0.0730
0.72	6840474	6.84	0.7250	0.7088	0.0629	0.7088	0.0629	0.5975	0.0631
0.76	7739007	7.74	0.7600	0.7367	0.0651	0.7367	0.0651	0.6248	0.0640
0.80	9257914	9.26	0.8000	0.7805	0.0428	0.7805	0.0428	0.6671	0.0433
0.84	12835873	12.84	0.8400	0.8261	0.0283	0.8261	0.0283	0.7124	0.0302
0.88	25599156	25.60	0.8530	0.8397	0.0170	0.8397	0.0170	0.7262	0.0197
Approximate Max Ao >>	> 25599156	> 25.60	0.8530	0.8454	0.0128	0.8454	0.0128	0.7316	0.0161

Table 2 Case Y (Variable) Output

4 Aircraft
2.0 hr TransitTime, 8.0 hr PatrolTime, 2.0 hr InspectTime
Variable MTTF, Constant MTTR, Constant OST

reps = 100

Goal Ao	Spares Budget	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0858	0.0041	0.0978	0.0044	0.0567	0.0058
0.04	1112889	1.11	0.0430	0.1759	0.0176	0.2002	0.0196	0.1165	0.0152
0.08	1699006	1.70	0.0810	0.2094	0.0216	0.2382	0.0239	0.1405	0.0183
0.12	2195323	2.20	0.1220	0.2324	0.0254	0.2638	0.0279	0.1549	0.0219
0.16	2694876	2.69	0.1710	0.2695	0.0284	0.3049	0.0313	0.1830	0.0235
0.20	3249008	3.25	0.2230	0.3082	0.0307	0.3475	0.0335	0.2123	0.0271
0.24	3444643	3.44	0.2430	0.3167	0.0319	0.3569	0.0348	0.2173	0.0280
0.28	3790885	3.79	0.2820	0.3364	0.0338	0.3785	0.0364	0.2325	0.0293
0.32	4171304	4.17	0.3210	0.3530	0.0383	0.3966	0.0414	0.2470	0.0333
0.36	4586503	4.59	0.3640	0.3844	0.0361	0.4305	0.0388	0.2707	0.0327
0.40	5199441	5.20	0.4190	0.4119	0.0413	0.4602	0.0440	0.2942	0.0374
0.44	5546911	5.55	0.4510	0.4380	0.0335	0.4881	0.0358	0.3149	0.0305
0.48	6158758	6.16	0.5030	0.4758	0.0356	0.5285	0.0375	0.3468	0.0331
0.52	6399776	6.40	0.5230	0.4878	0.0437	0.5413	0.0462	0.3570	0.0416
0.56	6974610	6.97	0.5650	0.5155	0.0424	0.5706	0.0444	0.3814	0.0399
0.60	7537554	7.54	0.6020	0.5395	0.0399	0.5958	0.0417	0.4025	0.0383
0.64	8444231	8.44	0.6440	0.5686	0.0440	0.6263	0.0457	0.4263	0.0413
0.68	9767942	9.77	0.6920	0.6145	0.0312	0.6740	0.0322	0.4691	0.0309
0.72	10700589	10.70	0.7200	0.6400	0.0288	0.7005	0.0296	0.4923	0.0293
0.76	13304881	13.30	0.7600	0.6767	0.0239	0.7382	0.0244	0.5263	0.0264
0.80	21770298	21.77	0.7870	0.6947	0.0181	0.7565	0.0184	0.5438	0.0221
Approximate Max Ao >>	> 21770298	> 21.77	0.7870	0.7045	0.0107	0.7665	0.0107	0.5527	0.0169

Table 3 Case Z (Constant) Output

4 Aircraft
2.0 hr TransitTime, 8.0 hr PatrolTime, 2.0 hr InspectTime
Variable MTTF, Variable MTTR, Variable OST

reps = 100

Goal Ao	Spares Budget	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0640	0.0084	0.0733	0.0096	0.0413	0.0069
0.04	1112889	1.11	0.0430	0.1644	0.0251	0.1876	0.0282	0.1072	0.0195
0.08	1699006	1.70	0.0810	0.2019	0.0293	0.2298	0.0328	0.1340	0.0227
0.12	2195323	2.20	0.1220	0.2337	0.0373	0.2652	0.0415	0.1570	0.0297
0.16	2694876	2.69	0.1710	0.2649	0.0394	0.2998	0.0433	0.1794	0.0314
0.20	3249008	3.25	0.2230	0.3114	0.0426	0.3508	0.0466	0.2144	0.0348
0.24	3444643	3.44	0.2430	0.3174	0.0403	0.3572	0.0439	0.2201	0.0340
0.28	3790885	3.79	0.2820	0.3388	0.0456	0.3806	0.0493	0.2362	0.0390
0.32	4171304	4.17	0.3210	0.3560	0.0453	0.3997	0.0490	0.2493	0.0387
0.36	4586503	4.59	0.3640	0.3914	0.0491	0.4375	0.0528	0.2791	0.0423
0.40	5199441	5.20	0.4190	0.4330	0.0527	0.4825	0.0561	0.3117	0.0468
0.44	5546911	5.55	0.4510	0.4507	0.0552	0.5013	0.0587	0.3258	0.0489
0.48	6158758	6.16	0.5030	0.4817	0.0523	0.5352	0.0553	0.3538	0.0473
0.52	6399776	6.40	0.5230	0.5008	0.0491	0.5546	0.0518	0.3689	0.0445
0.56	6974610	6.97	0.5650	0.5247	0.0527	0.5797	0.0551	0.3902	0.0488
0.60	7537554	7.54	0.6020	0.5535	0.0597	0.6099	0.0625	0.4162	0.0544
0.64	8444231	8.44	0.6440	0.5817	0.0509	0.6393	0.0530	0.4400	0.0480
0.68	9767942	9.77	0.6920	0.6221	0.0443	0.6816	0.0457	0.4765	0.0428
0.72	10700589	10.70	0.7200	0.6443	0.0416	0.7044	0.0428	0.4965	0.0420
0.76	13304881	13.30	0.7600	0.6775	0.0288	0.7385	0.0292	0.5279	0.0300
0.80	21770298	21.77	0.7870	0.6963	0.0275	0.7578	0.0280	0.5460	0.0296
Approximate Max Ao >>	> 21770298	> 21.77	0.7870	0.7064	0.0151	0.7680	0.0153	0.5564	0.0199

Table 4 Case Z (Variable) Output (Same as Base Case)

5 Aircraft
2.0 hr TransitTime, 8.0 hr PatrolTime, 2.0 hr InspectTime
Variable MTTF, Variable MTTR, Variable OST

reps = 100

Goal Ao	Spares Budget	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0673	0.0071	0.0767	0.0081	0.0443	0.0064
0.04	639182	0.64	0.0410	0.1326	0.0166	0.1507	0.0186	0.0891	0.0133
0.08	1256803	1.26	0.0860	0.1887	0.0252	0.2131	0.0278	0.1288	0.0204
0.12	1699006	1.70	0.1300	0.2079	0.0309	0.2341	0.0337	0.1439	0.0255
0.16	2083412	2.08	0.1700	0.2415	0.0354	0.2711	0.0383	0.1687	0.0298
0.20	2371048	2.37	0.2020	0.2525	0.0360	0.2833	0.0389	0.1779	0.0304
0.24	2694876	2.69	0.2410	0.2797	0.0370	0.3126	0.0398	0.1982	0.0317
0.28	3249008	3.25	0.2990	0.3283	0.0416	0.3647	0.0443	0.2388	0.0369
0.32	3444643	3.44	0.3210	0.3504	0.0446	0.3881	0.0472	0.2582	0.0404
0.36	3790885	3.79	0.3620	0.3580	0.0436	0.3968	0.0460	0.2638	0.0389
0.40	4149673	4.15	0.4000	0.3863	0.0500	0.4264	0.0526	0.2880	0.0452
0.44	4586503	4.59	0.4440	0.4243	0.0562	0.4663	0.0587	0.3215	0.0516
0.48	5199441	5.20	0.4970	0.4606	0.0610	0.5042	0.0634	0.3527	0.0568
0.52	5546911	5.55	0.5280	0.4859	0.0570	0.5307	0.0588	0.3770	0.0556
0.56	6158758	6.16	0.5760	0.5363	0.0591	0.5827	0.0607	0.4227	0.0565
0.60	6490552	6.49	0.6020	0.5569	0.0603	0.6038	0.0618	0.4418	0.0579
0.64	7071551	7.07	0.6400	0.5833	0.0608	0.6311	0.0622	0.4660	0.0586
0.68	8169614	8.17	0.6920	0.6333	0.0581	0.6818	0.0581	0.5144	0.0560
0.72	9071782	9.07	0.7240	0.6540	0.0516	0.7029	0.0523	0.5343	0.0517
0.76	10046927	10.05	0.7620	0.6981	0.0360	0.7476	0.0362	0.5764	0.0369
0.80	13041712	13.04	0.8000	0.7322	0.0324	0.7819	0.0326	0.6113	0.0327
0.84	23499064	23.50	0.8260	0.7564	0.0193	0.8062	0.1940	0.6355	0.0205
0.88	23499064	23.50	0.8260	0.7564	0.0193	0.8062	0.0194	0.6355	0.0205
Approximate Max Ao >>	> 23499064	> 23.50	0.8260	0.7625	0.0018	0.8123	0.0018	0.6391	0.0157

Table 5 Case A Output

4 Aircraft, 25% better MTBF, 50% increased cost
2.0 hr TransitTime, 8.0 hr PatrolTime, 2.0 hr InspectTime
Variable MTTF, Variable MTTR, Variable OST

reps = 100

Goal Ao	Spares Budget ($)	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0712	0.0103	0.0816	0.0118	0.0491	0.0092
0.04	868710	0.87	0.0400	0.1469	0.0242	0.1678	0.0273	0.1036	0.0206
0.08	1704521	1.70	0.0830	0.2108	0.0329	0.2395	0.0366	0.1513	0.0275
0.12	2258717	2.26	0.1240	0.2310	0.0337	0.2620	0.0373	0.1671	0.0289
0.16	2863505	2.86	0.1730	0.2627	0.0410	0.2972	0.0450	0.1917	0.0342
0.20	3192363	3.19	0.2010	0.2841	0.0392	0.3207	0.0430	0.2085	0.0340
0.24	3670561	3.67	0.2420	0.3147	0.0526	0.3542	0.0568	0.2343	0.0462
0.28	4634712	4.63	0.3190	0.3684	0.0531	0.4125	0.0572	0.2768	0.0474
0.32	4697514	4.70	0.3250	0.3574	0.0544	0.4009	0.0585	0.2680	0.0479
0.36	5198752	5.20	0.3670	0.3926	0.0604	0.4385	0.0648	0.2990	0.0537
0.40	5702889	5.70	0.4040	0.4189	0.0590	0.4669	0.0632	0.3212	0.0524
0.44	6394231	6.39	0.4530	0.4419	0.0648	0.4915	0.0690	0.3403	0.0584
0.48	6787275	6.79	0.4810	0.4589	0.0623	0.5099	0.0659	0.3547	0.0588
0.52	7446194	7.45	0.5210	0.4980	0.0577	0.5509	0.0608	0.3901	0.0552
0.56	8556676	8.56	0.5830	0.5457	0.0560	0.6014	0.0587	0.4313	0.0526
0.60	9019317	9.02	0.6050	0.5500	0.0633	0.6057	0.0662	0.4350	0.0598
0.64	9941782	9.94	0.6470	0.5930	0.0551	0.6507	0.0575	0.4748	0.0522
0.68	10771041	10.77	0.6810	0.6144	0.0623	0.6728	0.0646	0.4936	0.0599
0.72	12659163	12.66	0.7280	0.6515	0.0464	0.7114	0.0478	0.5284	0.0455
0.76	14346116	14.35	0.7620	0.6915	0.0373	0.7523	0.0382	0.5671	0.0369
0.80	17831276	17.83	0.8010	0.7220	0.0290	0.7835	0.0295	0.5956	0.0315
0.82	23393858	23.39	0.8210	0.7434	0.0203	0.8052	0.0205	0.6158	0.0231
Approximate Max Ao >>	>23393858	> 23.39	0.8250	0.7520	0.0125	0.8140	0.0125	0.6252	0.0167

Table 6 Case B Output

4 Aircraft
2.0 hr TransitTime, 8.0 hr PatrolTime, 2.0 hr InspectTime
Variable MTTF (Weibull), Variable MTTR, Variable OST

reps = 100

Goal Ao	Spares Budget	Spares Budget ($M)	ARROWs Ao	SIMACE mu Ao	SIMACE sigma Ao	SIMACE mu FMC	SIMACE sigma FMC	SIMACE mu otFMC	SIMACE sigma otFMC
0.00	0	0.00	0.0000	0.0837	0.0102	0.0957	0.0116	0.0629	0.0097
0.04	1112889	1.11	0.0430	0.2191	0.0337	0.2484	0.0372	0.1615	0.0286
0.08	1699006	1.70	0.0810	0.2740	0.0376	0.3089	0.0411	0.2043	0.0316
0.12	2195323	2.20	0.1220	0.3210	0.0431	0.3604	0.0467	0.2404	0.0378
0.16	2694876	2.69	0.1710	0.3489	0.0475	0.3909	0.0514	0.2625	0.0410
0.20	3249008	3.25	0.2230	0.4090	0.0597	0.4554	0.0637	0.3134	0.0537
0.24	3444643	3.44	0.2430	0.4216	0.0488	0.4693	0.0520	0.3229	0.0437
0.28	3790885	3.79	0.2820	0.4392	0.0571	0.4880	0.0607	0.3375	0.0506
0.32	4171304	4.17	0.3210	0.4654	0.0518	0.5162	0.0547	0.3607	0.0466
0.36	4586503	4.59	0.3640	0.4964	0.0561	0.5487	0.0592	0.3867	0.0513
0.40	5199441	5.20	0.4190	0.5278	0.0637	0.5820	0.0667	0.4138	0.0591
0.44	5546911	5.55	0.4510	0.5484	0.0566	0.6036	0.0592	0.4323	0.0520
0.48	6158758	6.16	0.5030	0.5880	0.0591	0.6452	0.0617	0.4676	0.0548
0.52	6399776	6.40	0.5230	0.5973	0.0594	0.6549	0.0617	0.4758	0.0568
0.56	6974610	6.97	0.5650	0.6282	0.0560	0.6871	0.0579	0.5034	0.0546
0.60	7537554	7.54	0.6020	0.6448	0.0524	0.7042	0.0541	0.5194	0.0516
0.64	8444231	8.44	0.6440	0.6675	0.0437	0.7278	0.0448	0.5394	0.0430
0.68	9767942	9.77	0.6920	0.7049	0.0360	0.7661	0.0369	0.5751	0.0367
0.72	10700589	10.70	0.7200	0.7211	0.0247	0.7826	0.0251	0.5913	0.0269
0.76	13304881	13.30	0.7600	0.7312	0.0218	0.7928	0.0222	0.6002	0.0255
0.80	21770298	21.77	0.7870	0.7453	0.0133	0.8072	0.0134	0.6138	0.0177
Approximate Max Ao >>	> 21770298	> 21.77	0.7870	0.7459	0.0120	0.8079	0.0121	0.6146	0.0167

Table 7 Application of the Weibull Distribution to MTTF

APPENDIX D. SIMACE JAVA CODE

The purpose of this appendix is to display the Java code used to develop SIMACE. The Java code begins on the next page.

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: Main class for SIMACE.
 */

import simkit.*;
import simkit.random.*;
import simkit.util.*;
import simkit.stat.*;
import simkit.xml.*;

import java.util.*;
import java.text.*;
import java.io.*;
import java.net.*;

import javax.swing.*;
import javax.swing.filechooser.*;

import org.jdom.*;
import org.jdom.input.SAXBuilder;

public class ACE_main {

   // basic information

   private static int numAircraft;
   private static double transitTime;
   private static double patrolTime;
   private static double inspectionTime;
   private static double deploymentLength;

   // information for each WRA

   private static String label;
   private static String nomenclature;
   private static double unitPrice;
   private static int qtyPerAircraft;
   private static int qtySpare;

   private static String ttfDistribution;
   private static double ttfValue;
   private static int ttfSeed;

   private static String ttrDistribution;
   private static double ttrValue;
   private static int ttrSeed;

   private static String ostDistribution;
   private static double ostValue;
   private static int ostSeed;
```

```
private static RandomVariate ttfRV;
private static RandomVariate ttrRV;
private static RandomVariate ostRV;

// the containers

private static Aircraft[] squadron;
private static HashMap wraMap = new HashMap();
private static HashMap ttfMap = new HashMap();
private static HashMap ttrMap = new HashMap();
private static HashMap ostMap = new HashMap();
private static HashMap acAllowanceMap = new HashMap();
private static HashMap spareAllowanceMap = new HashMap();
private static WRAInventory spareInv = new WRAInventory();

// the ATO object

private static ATO ato;

// formatting variables

private static DecimalFormat fmt = new DecimalFormat("0.0000");

// dumper, for observing the property changes on the event list

private static SimplePropertyDumper dumper = new SimplePropertyDumper();

// statistical variables

private static SimpleStatsTimeVarying otStat
   = new SimpleStatsTimeVarying("operational");
private static SimpleStatsTimeVarying otFMCStat
   = new SimpleStatsTimeVarying("operationalFMC");
private static SimpleStatsTimeVarying stStat
   = new SimpleStatsTimeVarying("standBy");
private static SimpleStatsTimeVarying fmcStat
   = new SimpleStatsTimeVarying("fmc");
private static SimpleStatsTimeVarying inspectStat
   = new SimpleStatsTimeVarying("inspecting");

private static SimpleStatsTally aoTally
   = new SimpleStatsTally();
private static SimpleStatsTally fmcTally
   = new SimpleStatsTally();
private static SimpleStatsTally inFlightFailureFMCTally
   = new SimpleStatsTally();

private static File outFile;
private static File outFileCSV;

protected static RandomNumber ttfRNG;
protected static RandomNumber ttrRNG;
protected static RandomNumber ostRNG;

// main method
```

```
public static void main(String args[]) throws IOException {

   /**
    * Time to Failure
    * There is a 1-1 correspondance between each individual WRA and ttf.
    * Each WRA has its own ttf RandomVariate.  The TTF Random Variates
    * are stored in a HashMap.  When instantiating a WRA, the appropriate
    * ttf RandomVariate is accessed from this HashMap.  The HashMap is
    * referenced when individual Aircraft are instantiated. This HashMap
    * will NOT be sent to the ATO class.
    *
    * Time to Repair
    * There is NOT a 1-1 correspondance between individual, independent
    * WRAs and Time to Repair (ttr).  Rather a relationship exists
    * between all WRAs of a particular type (label) and the ttr. For
    * example, all WRAs with a label of "000123456" have the same
    * ttr RandomVariate whereas each INDIVIDUAL "000123456" has its
    * own ttf RandomVariate.  The TTR RandomVariates are stored in a
    * HashMap.  This HashMap will be sent to the ATO class.
    *
    * Ordering and Shipping Time
    * There is NOT a 1-1 correspondance between individual, independent
    * WRAs and Ordering and Shipping Time (ost).  Rather a relationship
    * exists between all WRAs of a particular type (label) and the ost
    * RandomVariate.  For example all WRAs with a label of "000123456"
    * have the same ost distribution whereas each INDIVIDUAL "000123456"
    * has its own ttf RandomVariate.  The OST Random Variates are stored
    * in a HashMap. This HashMap will be sent to the ATO class.
    *
    * The Squadron of Aircraft
    * A squadron of Aircraft consists of a certain quantity of Aircraft.
    * Aircraft are referenced by their tail number.  The tail numbers
    * are always strings of two digits.  For example "00", "07", and "11".
    * The squadron of Aircraft is an Array of type Aircraft.  The
    * squadron Array is sent to the ATO class.
    *
    * WRAs
    * The number of each type WRA that belong in each Aircraft object is
    * an input parameter.  Each Aircraft has a certain number of each
    * type WRA installed on it.  The possibility exists that an Aircraft
    * can have multiple instances of the same type WRA.  For example, an
    * Aircraft can have more than one WRA with a label of "000123456".
    * WRAs are always referenced referenced by their label which must
    * be unique for each type WRA.
    *
    * The Spares Kit
    * The allowance for each WRA that makes up the spares kit is an
    * input parameter.  The spares kit is stored as a WRAInventory.
    * The  spares kit WRAInventory is sent to the ATO class.
    **/

   // Select output file for data archiving (collects data for EACH run)

   URL url = ACE_main.class.getResource("ACE_main.class");
   File outDirectory = new File( url.getFile() ).getParentFile();
   JFileChooser outChooser = new JFileChooser(outDirectory);
   int outResult = outChooser.showDialog(null, "Save");
```

```
   if( outResult != JFileChooser.APPROVE_OPTION) {
      System.err.println("No save file chosen.");
      System.exit(0);
   }
   outFile = outChooser.getSelectedFile();

   // Select output file for entry into a spreadsheet

   URL urlCSV = ACE_main.class.getResource("ACE_main.class");
   File outDirectoryCSV = new File( urlCSV.getFile() ).getParentFile();
   JFileChooser outChooserCSV = new JFileChooser(outDirectoryCSV);
   int outResultCSV = outChooserCSV.showDialog(null, "Save");
   if( outResultCSV != JFileChooser.APPROVE_OPTION) {
      System.err.println("No save file chosen.");
      System.exit(0);
   }
   outFileCSV = outChooserCSV.getSelectedFile();

   // Select input files

   JFileChooser inChooser
         = new JFileChooser(ACE_main.class.getResource("ACE_main.class")
         .getFile());
   int inResult = inChooser.showDialog(new JFrame(), "Open");
   if(inResult != JFileChooser.APPROVE_OPTION) {
      System.err.println("No input file selected...exiting.");
      System.exit(0);
   }
   File inFile = inChooser.getSelectedFile();
   File inDirectory = inFile.getParentFile();
   File[] files = inDirectory.listFiles();

   for( int i = 0; i < files.length; ++i ) {
      processOneScenario( files[i] );
      Schedule.clearRerun();
      resetScenario();
   }
   System.out.println( "SIMACE completed its processing.
   Please see the output file." );
   System.exit(0);
}

public static void processOneScenario(File inputFile) throws IOException {
   double nextAo;
   double nextFMC;
   double nextInFlightFailureFMC;
   String string = "";
   FileOutputStream out = new FileOutputStream(outFile, true);
   FileOutputStream outCSV = new FileOutputStream(outFileCSV, true);

   string = "\n**********************************************************";
   out.write(string.getBytes());
   string = "\nProcessing: " + inputFile;
   out.write(string.getBytes());
   string = "\n**********************************************************";
   out.write(string.getBytes());
   try {
```

```
   processXMLFiles(inputFile);
}
catch (FileNotFoundException e) {
   System.out.println("FileNotFoundException");
}
createSquadron();
passParameters();
addListeners();

string = "\n\nFormat: nextAo, nextFMC, nextInFlightFailureFMC";
out.write(string.getBytes());

for( int i = 0; i < 100; ++i ) {
   Schedule.reset();
   resetTimeVaryingStats();
   populateAircraft();
   populateSparesKit();
   runSimulation();

   nextAo = calculateAo();
   aoTally.newObservation( nextAo );
   string = "\n" + nextAo + ",";
   out.write(string.getBytes());

   nextFMC = calculateFMC();
   fmcTally.newObservation( nextFMC );
   string = "" + nextFMC + ",";
   out.write(string.getBytes());

   nextInFlightFailureFMC = calculateInFlightFailureFMC();
   inFlightFailureFMCTally.newObservation( nextInFlightFailureFMC );
   string = "" + nextInFlightFailureFMC;
   out.write(string.getBytes());
}

string = "\n\naoTally.getCount()=" + aoTally.getCount();
out.write(string.getBytes());
string = "\naoTally.getMean()=" + fmt.format( aoTally.getMean() ) ;
out.write(string.getBytes());
string = "," + aoTally.getMean();
outCSV.write(string.getBytes());
string = "\naoTally.getStandardDeviation()="
      + fmt.format( aoTally.getStandardDeviation() );
out.write(string.getBytes());
string = "," + aoTally.getStandardDeviation();
outCSV.write(string.getBytes());

string = "\n\nfmcTally.getCount()=" + fmcTally.getCount();
out.write(string.getBytes());
string = "\nfmcTally.getMean()=" + fmt.format( fmcTally.getMean() );
out.write(string.getBytes());
string = "," + fmcTally.getMean();
outCSV.write(string.getBytes());
string = "\nfmcTally.getStandardDeviation()="
      + fmt.format( fmcTally.getStandardDeviation() );
out.write(string.getBytes());
string = "," + fmcTally.getStandardDeviation();
```

```
   outCSV.write(string.getBytes());

   string = "\n\ninFlightFailureFMCTally.getCount()=" +
         inFlightFailureFMCTally.getCount();
   out.write(string.getBytes());
   string = "\ninFlightFailureFMCTally.getMean()=" +
         fmt.format( inFlightFailureFMCTally.getMean() );
   out.write(string.getBytes());
   string = "," + inFlightFailureFMCTally.getMean();
   outCSV.write(string.getBytes());
   string = "\ninFlightFailureFMCTally.getStandardDeviation()=" +
         fmt.format( inFlightFailureFMCTally.getStandardDeviation() );
   out.write(string.getBytes());
   string = "," + inFlightFailureFMCTally.getStandardDeviation();
   outCSV.write(string.getBytes());

   string = "\n\n";
   out.write(string.getBytes());
   out.close();
   outCSV.close();
}

/**
 * Calculates and returns the Ao for a simulation run.
 **/
public static double calculateAo() {
   return (otStat.getMean() + stStat.getMean())/numAircraft;
}

/**
 * Calculates and returns the FMC rate for a simulation run.
 **/
public static double calculateFMC() {
   return (otStat.getMean() + stStat.getMean() +
         inspectStat.getMean())/numAircraft;
}

/**
 * Calculates the FMC rate, not including the time operational
 * with a NRFI WRA.  It could be the case that one or more
 * WRAs fail in flight.  If this is the case, this method
 * calculates the FMC rate NOT including the operational time
 * after the first WRA failure.
 **/
public static double calculateInFlightFailureFMC() {
   return (otFMCStat.getMean() + stStat.getMean() +
    inspectStat.getMean())/numAircraft;
}

/**
 * Creates a squadron of Aircraft.  The Aircraft are populated
 * with WRAs in the method populateAircraft().
 **/
public static void createSquadron() {
   String tailNumber;
   for( int i = 0; i < squadron.length; ++i ) {
      if( i < 10 ) {
```

```
            tailNumber = "0" + String.valueOf(i);
         }
         else {
            tailNumber = String.valueOf(i);
         }
         squadron[i] = new Aircraft( tailNumber );
      }
   }

   /**
    * Populate each aircraft with the proper allowance of each type WRA.
    **/
   public static void populateAircraft() {
      String label;
      Set labels = acAllowanceMap.keySet();
      WRA wra;
      int allowance = 0;
      for(Iterator iter = labels.iterator(); iter.hasNext();) {
         label = (String)iter.next();
         allowance = ( (Integer)acAllowanceMap.get(label) ).intValue();
         wra = new WRA( (WRA)wraMap.get(label) );

         for(int i = 0; i < squadron.length; ++i) {
            squadron[i].addWRA(wra, allowance);
         }
      }
   }

   /**
    * Populate the spares kit with the proper allowance of each type WRA.
    **/
   public static void populateSparesKit() {
      String label;
      Set labels = spareAllowanceMap.keySet();
      WRA wra;
      int allowance = 0;
      for(Iterator iter = labels.iterator(); iter.hasNext();) {
         label = (String)iter.next();
         allowance = ( (Integer)spareAllowanceMap.get(label) ).intValue();
         wra = new WRA( (WRA)wraMap.get(label) );
         spareInv.addCopies(wra, allowance);
      }
   }

   /**
    * Pass the parameters to the appropriate classes.
    **/
   public static void passParameters() {
      Object[] params = new Object[8];
      params[0] = squadron;
      params[1] = spareInv;
      params[2] = wraMap;
      params[3] = ttrMap;
      params[4] = ostMap;
      params[5] = new Double(transitTime);
      params[6] = new Double(patrolTime);
      params[7] = new Double(inspectionTime);
```

```
   ato = new ATO(params);
}

/**
 * Add listeners.
 **/
public static void addListeners() {

   // If desired, dumpers may listen for convience.

   // for( int i = 0; i < squadron.length; i++ ) {
   //   squadron[i].addPropertyChangeListener( dumper );
   // }
   // ato.addPropertyChangeListener( dumper );

   for( int i = 0; i < squadron.length; ++i ) {
      squadron[i].addSimEventListener( ato );
   }

   // ato.addPropertyChangeListener( dumper );

   ato.addPropertyChangeListener( otStat );
   ato.addPropertyChangeListener( otFMCStat );
   ato.addPropertyChangeListener( stStat );
   ato.addPropertyChangeListener( fmcStat );
   ato.addPropertyChangeListener( inspectStat );
}

/**
 * Run the simulation.
 **/
public static void runSimulation() {
   Schedule.stopAtTime( deploymentLength );
   Schedule.setVerbose( false );
   Schedule.setSingleStep( false );
   Schedule.startSimulation();
}

public static void processXMLFiles(File inputFile) throws
   FileNotFoundException, IOException {

   SAXBuilder builder;
   Document doc = null;
   FileOutputStream out = new FileOutputStream(outFile, true);
   FileOutputStream outCSV = new FileOutputStream(outFileCSV, true);

   String string = "";
   try {
      builder = new SAXBuilder();
      doc = builder.build( inputFile );
   }
   catch (JDOMException e) {}
   catch (Exception e) {}
   Element root = doc.getRootElement();
   if( root.getChild("NumAircraft") != null ) {

      string = "\n" + root.getChild("AvailabilityGoal").getText();
```

```
outCSV.write(string.getBytes());
string = "," + root.getChild("SparesCost").getText();
outCSV.write(string.getBytes());
string = "," +
   root.getChild("SparesEstimatedAvailability").getText();
outCSV.write(string.getBytes());

Element numAircraftElement = root.getChild("NumAircraft");
numAircraft = Integer.parseInt( numAircraftElement.getText() );
squadron = new Aircraft[numAircraft];
string = "\nnumAircraft = " + numAircraft;
out.write(string.getBytes());

Element transitTimeElement = root.getChild("TransitTime");
transitTime = Double.parseDouble( transitTimeElement.getText() );
string = "\ntransitTime = " + fmt.format(transitTime);
out.write(string.getBytes());

Element patrolTimeElement = root.getChild("PatrolTime");
patrolTime = Double.parseDouble( patrolTimeElement.getText() );
string = "\npatrolTime = " + fmt.format(patrolTime);
out.write(string.getBytes());

Element inspectionTimeElement = root.getChild("InspectionTime");
inspectionTime = Double.parseDouble( inspectionTimeElement.
   getText() );
string = "\ninspectionTime = " + fmt.format(inspectionTime);
out.write(string.getBytes());

Element deploymentLengthElement = root.getChild("DeploymentLength");
deploymentLength = Double.parseDouble( deploymentLengthElement
   .getText() );
string = "\ndeploymentLength = " + fmt.format(deploymentLength);
out.write(string.getBytes());

Element availabilityGoalElement = root.getChild("AvailabilityGoal");
string = "\n\nAvailabilityGoal = " +
   availabilityGoalElement.getText();
out.write(string.getBytes());

Element sparesCostElement = root.getChild("SparesCost");
string = "\nSparesCost = " + sparesCostElement.getText();
out.write(string.getBytes());

Element sparesEstimatedAvailabilityElement
   = root.getChild("SparesEstimatedAvailability");
string = "\nSparesEstimatedAvailability = " +
   sparesEstimatedAvailabilityElement.getText();
out.write(string.getBytes());

ttfRNG = createTTFRNG( root.getChild("MTTFSeed") );
ttrRNG = createTTRRNG( root.getChild("MTTRSeed") );
ostRNG = createOSTRNG( root.getChild("OSTSeed") );

Element wraElement;
WRA wra;
List wraElements = root.getChild("WRAData").getChildren("WRA");
```

```
      string = "\nThere are " + wraElements.size() +
         " WRA elements in the XML file.";
      out.write(string.getBytes());

      for( int i = 0; i < wraElements.size(); i++ ) {
         wraElement = (Element)wraElements.get(i);
         label = wraElement.getChild("Label").getText();
         nomenclature = wraElement.getChild("Nomenclature").getText();
         unitPrice = Double.parseDouble(wraElement.
         getChild("UnitPrice").getText());
         qtyPerAircraft = Integer.parseInt(wraElement.
         getChild("QtyPerAircraft").getText());
         qtySpare = Integer.parseInt(wraElement.
         getChild("QtySpare").getText());

         ttfDistribution = wraElement.
         getChild("MTTFDistribution").getText();
         ttfValue = Double.parseDouble(wraElement.
         getChild("MTTFValue").getText());

         ttrDistribution = wraElement.
         getChild("MTTRDistribution").getText();
         ttrValue = Double.parseDouble(wraElement.
         getChild("MTTRValue").getText());

         ostDistribution = wraElement.
         getChild("OSTDistribution").getText();
         ostValue = Double.parseDouble(wraElement.
         getChild("OSTValue").getText());

         ttfRV = RandomVariateFactory.getInstance( ttfDistribution,
               new Object[] {new Double(ttfValue)}, ttfRNG );
         ttrRV = RandomVariateFactory.getInstance( ttrDistribution,
               new Object[] {new Double(ttrValue)}, ttrRNG );
         ostRV = RandomVariateFactory.getInstance( ostDistribution,
               new Object[] {new Double(ostValue)}, ostRNG );
         ttfMap.put(label, ttfRV);
         ttrMap.put(label, ttrRV);
         ostMap.put(label, ostRV);

         wra = new WRA(label, nomenclature, ttfRV);
         wraMap.put(label, wra);
         acAllowanceMap.put(label, new Integer(qtyPerAircraft));
         spareAllowanceMap.put(label, new Integer(qtySpare));
         out.close();
         outCSV.close();
      }
      System.out.println( "Done processing the file: " + inputFile );
   }
}

public static RandomNumber createTTFRNG( Element ttfSeedElement ) {
   int seed = Integer.parseInt( ttfSeedElement.getText() );
   return RandomNumberFactory.getInstance( CongruentialSeeds.SEED[seed] );
}

public static RandomNumber createTTRRNG( Element ttrSeedElement ) {
```

```
      int seed = Integer.parseInt( ttrSeedElement.getText() );
      return RandomNumberFactory.getInstance( CongruentialSeeds.SEED[seed] );
   }

   public static RandomNumber createOSTRNG( Element ostSeedElement ) {
      int seed = Integer.parseInt( ostSeedElement.getText() );
      return RandomNumberFactory.getInstance( CongruentialSeeds.SEED[seed] );
   }

   public static void resetTimeVaryingStats() {
      otStat.reset();
      otFMCStat.reset();
      stStat.reset();
      fmcStat.reset();
      inspectStat.reset();
   }

   public static void resetScenario() {
      wraMap.clear();
      ttfMap.clear();
      ttrMap.clear();
      ostMap.clear();
      acAllowanceMap.clear();
      spareAllowanceMap.clear();
      spareInv.clear();
      ato = null;

      aoTally.reset();
      fmcTally.reset();
      inFlightFailureFMCTally.reset();
   }
}
```

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: The "meat" of SIMACE.  Coordinates the take-off, landing,
 * supply requisitioning, and maintenance processes.
 */

import simkit.*;
import simkit.random.*;
import simkit.util.*;
import simkit.stat.*;
import java.util.*;
import java.text.DecimalFormat;
import exceptions.*;

public class ATO extends SimEntityBase {

   // instance variables

   private HashMap ttrMap;        // time to repair RandomVariates
   private HashMap ostMap;        // order and shipping time RandomVariates
   private DecimalFormat fmt;     // formats numbers
   private Comparator wraComp;    // comparator for the WRA AWP lists
   private Comparator readyComp;  // comparator for the list of ready Aircraft
   private double transitTime;    // set inside the main
   private double patrolTime;     // set inside the main
   private double inspectionTime; // set inside the main

   // state variables

   protected Aircraft[] originalSquadron;
   protected HashMap originalWRAMap;

   protected ArrayList ready;         // the list of "ready to go" Aircraft
   protected HashMap nmcMap;          // NMC Aircraft
   protected HashMap awpQueues;       // for sorting aircraft by AWP WRAs
   protected WRAInventory spareInv;   // inventory of spares
   protected int numStandBy;          // the number of Aircraft currently
                                      // on stand by
   protected int numOperational;      // the number of Aircraft currently
                                      // operational
   protected int numOperationalFMC;   // the number of Aircraft currently
                                      // operational and FMC
   protected int numInspect;          // the number of Aircraft currently
                                      // being inspected
   protected int numFMC;              // the number of Aircraft currently FMC
   protected boolean needed;          // true when an Aircraft is needed
                                      // for a sortie

   // constructor methods

   /**
```

```
 * Reads in an array of Objects.  If an array is not read in,
 * the only other way to read in the parameters would be an
 * extremely long list of parameters, which would be very ugly.
 **/
public ATO(Object[] params) {
   ready = new ArrayList();
   nmcMap = new HashMap();
   ttrMap = new HashMap();
   ostMap = new HashMap();
   awpQueues = new HashMap();
   spareInv = new WRAInventory();
   fmt = new DecimalFormat( "0.00" );
   wraComp = new AircraftWRAComparator();
   readyComp = new AircraftReadyComparator();

   originalSquadron = (Aircraft[])params[0];

   // params[1] = the spareInv, a WRAInventory
   setSpareInv( (WRAInventory) params[1] );

   // params[2] = a copy (not clone) of the original list of WRAs
   originalWRAMap = (HashMap)params[2];

   // params[3] = HashMap of WRA TTR, key = WRA label
   setTTR( (HashMap) params[3] );

   // params[4] = HashMap of WRA OST, key = WRA label
   setOST( (HashMap) params[4] );

   // params[5] = transitTime, a double
   setTransitTime( ( (Double) params[5] ).doubleValue() );

   // params[6] = patrolTime, a double
   setPatrolTime( ( (Double) params[6] ).doubleValue() );

   // params[7] = inspectionTime, a double
   setInspectionTime( ( (Double) params[7] ).doubleValue() );

   setNeeded(true);
}

// instance methods

/**
 * How the simulation gets going (jump started).
 **/
public void doRun() {
   Aircraft aircraft;
   firePropertyChange( "standBy", 0, getNumStandBy() );
   setNeeded(false);
   waitDelay( "TakeOff", 0.0, removeNextReady() );
}

/**
 * An Aircraft takes off (starts its sortie) and then starts its
 * patrol.  The patrol is scheduled to begin after the Aircraft
 * transits to its patrol area (on station).  Another Aircraft will
```

```
 * be need to be on station before the one that is currently on patrol
 * departs station to return to base.  So, another Aircraft will
 * be needed in getPatrolTime().  Property changes are fired here
 * because the Aircraft does not become operational until it
 * actually takes-off.
 **/
public void doTakeOff( Aircraft aircraft ) {
   aircraft.startSortie();
   aircraft.startSortie( getSortieTime() );
   firePropertyChange( "standBy", numStandBy, --numStandBy );
   firePropertyChange( "operational", numOperational, ++numOperational );
   firePropertyChange( "operationalFMC", numOperationalFMC,
         ++numOperationalFMC );
   waitDelay( "StartPatrol", getTransitTime(), aircraft );
   waitDelay( "NeedsAircraft", getPatrolTime(), 1.0 );

}

/**
 * This event tells the system that an Aircraft isNeeded().
 **/
public void doNeedsAircraft() {
   setNeeded( true );
   if( getNumReady() > 0 ) {
      setNeeded( false );
      waitDelay( "TakeOff", 0.0, removeNextReady() );
   }
}

/**
 * An Aircraft starts a patrol and its end patrol time is
 * scheduled.
 **/
public void doStartPatrol( Aircraft aircraft ) {
   waitDelay( "EndPatrol", getPatrolTime(), aircraft );
}

/**
 * An Aircraft ends its patrol and its landing time
 * is scheduled.
 **/
public void doEndPatrol( Aircraft aircraft ) {
   waitDelay( "Land", getTransitTime(), aircraft );
}

/**
 * An Aircraft lands and begins its post-flight inspection process
 * is scheduled.  This simulation sets the post-flight inspection
 * to immediately begin after landing.  Of course, this can
 * be changed by anyone in the future.
 **/
public void doLand( Aircraft aircraft ) {
   aircraft.endSortie();
   firePropertyChange( "operational", numOperational, --numOperational );
   if( !aircraft.getInFlightFailure() ) {
      firePropertyChange( "operationalFMC", numOperationalFMC,
         --numOperationalFMC );
```

```
   }
   waitDelay( "StartInspect", 0.0, aircraft );
}

/**
 * An Aircraft begins its post-flight inspection.  The end inspection
 * time is then scheduled.
 **/
public void doStartInspect( Aircraft aircraft ) {
   firePropertyChange( "inspecting", numInspect, ++numInspect );
   waitDelay( "EndInspect", getInspectionTime(), aircraft );
}

/**
 * An Aircraft ends the post-flight inspection process.
 * If the Aircraft isFMC() it is returned to the list
 * of ready Aircraft.  If the Aircraft is !isFMC(),
 * this means that it has failed WRAs and must begin
 * the StartAircraftRepair process.
 **/
public void doEndInspect( Aircraft aircraft ) {
   firePropertyChange( "inspecting", numInspect, --numInspect );
   if( aircraft.isFMC() ) {
      waitDelay( "AddToReady", 0.0, aircraft );
   }
   else {
      addToNMC( aircraft );
      waitDelay( "StartAircraftRepair", 0.0, aircraft );
   }
}

/**
 * Places an Aircraft into the list of Aircraft ready
 * for a sortie.  If an Aircraft isNeeded(), then the
 * Aircraft most recently added to the list of ready
 * Aircraft is scheduled to immediately take off.
 **/
public void doAddToReady( Aircraft aircraft ) {
   addToReady( aircraft );
   firePropertyChange( "standBy", numStandBy, ++numStandBy );
   if( isNeeded() ) {
      setNeeded( false );
      waitDelay( "TakeOff", 0.0, removeNextReady() );
   }
}

/**
 * Starts the Aircraft repair process.  Since an Aircraft is starting the
 * repair process, it must have failed WRAs that were determined so during
 * inspection process.  Each of the failed WRAs are checked for availability
 * in the inventory of spares.  If a spare WRA exists, the spare is removed
 * from the inventory of spares, is placed in work, and the installation
 * process begins.  The spare WRA taken from the inventory of spares
 * must be replentished so a replacement WRA must be  ordered.  If a
 * spare WRA matching that of the failed WRA does not exist inside
 * the spares kit (inventory) it must be ordered and until the WRA
 * is received for the Aircraft in need the Aircraft in need is
```

```
 * considered awaiting parts (AWP) for this WRA.
 **/
public void doStartAircraftRepair( Aircraft aircraft ) {
   WRA spareWRA;
   WRA newWRA;
   double totalTTR = 0.0;

   while( aircraft.hasFailedWRAS() ) {
      WRA failedWRA = aircraft.removeFailedWRA();
      if( getNumberOfSpare( failedWRA ) > 0 ) {
         spareWRA = getSpare( failedWRA );
         totalTTR += getNextTTR( spareWRA );
         aircraft.addInWork( spareWRA );
         newWRA = new WRA( failedWRA );
         waitDelay( "StartOST", 0.0, newWRA );
      }
      else {
         aircraft.addAWP( failedWRA );
         addToAWPQueues( failedWRA );
         newWRA = new WRA( failedWRA );
         waitDelay( "StartOST", 0.0, newWRA );
      }
   }
   if( totalTTR > 0.0 ) {
      waitDelay( "EndAircraftRepair", totalTTR, aircraft );
   }
}

/**
 * An Aircraft ends the repair process.  If the Aircraft isFMC() after
 * it finishes the repair process, it is removed from the HashMap of
 * NMC aircraft and immediately added to the Array of ready Aircraft.
 * The case could exist that upon the completion of a repair, a WRA
 * an Aircraft was AWP for was delivered.  This means that there
 * is additional work to be done.  So if the Aircraft has more WRAs to
 * be installed, these WRAs are transferred from its "moreToDo" list
 * to its "inWork" list.
 **/
public void doEndAircraftRepair( Aircraft aircraft ) {
   aircraft.transferInWorkToAircraft();
   if( aircraft.isFMC() ) {
      removeFromNMC( aircraft );
      firePropertyChange("fmc", numFMC, ++numFMC);
      waitDelay( "AddToReady", 0.0, aircraft );
   }
   else if( aircraft.hasMoreToDo() ) {
      double additionalTTR = transferMoreToDo(aircraft);
      waitDelay( "EndAircraftRepair", additionalTTR, aircraft );
   }
}

/**
 * Starts the ordering and shipping process to replace a failed WRA.
 * The ordering and shipping time (ost) is generated and the new WRA is
 * ordered.  This simulation uses only one OST for all WRAs.  This is not
 * how things work in reality.  In the real world, there is hi-priority
 * and routine shipping, each of which has a different value.
```

```
 **/
public void doStartOST( WRA newWRA ) {
   String label = newWRA.getLabel();
   double timeForOST = ( (RandomVariate)ostMap.get(label) ).generate();
   waitDelay( "EndOST", timeForOST, newWRA );
}

/**
 * A WRA arrives from the ordering and shipping process.  Before being
 * installed, the Aircraft most in need for this WRA must be determined.
 * If none of the Aircraft are in need of the inbound WRA, it is
 * considered ordered as a replentishment item and is therefore
 * placed (returned) into the inventory of spares.
 **/
public void doEndOST(WRA newWRA) {
   assignNeedyAircraft( newWRA );
   Aircraft aircraft = newWRA.getBelongsTo();
   if( aircraft == null ) {
      addSpare( newWRA );
   }
   else {
      if( aircraft.hasInWork() ) {
         aircraft.removeAWP(newWRA);
         aircraft.addMoreToDo(newWRA);
      }
      else {
         aircraft.removeAWP(newWRA);
         aircraft.addInWork(newWRA);
         waitDelay( "EndAircraftRepair", getNextTTR(newWRA), aircraft );
      }
   }
}

/**
 * Listens for an InFlightWRAFailure in class Aircraft.
 **/
public void doInFlightWRAFailure() {
   firePropertyChange("operationalFMC", numOperationalFMC,
         --numOperationalFMC);
}

/**
 * Returns the Aircraft that is most in need of the specified WRA.
 * Only those Aircraft non-mission capable (NMC) are considered
 * eligible for the WRA.  The priority goes to the first aircraft in
 * the NMC queue that only needs this WRA to become FMC.  If none of
 * the Aircraft in the NMC queue only need this WRA to become FMC,
 * the WRA goes to the Aircraft with the most NMC time.  If there is
 * a tie for the above two sorting criteria, then the WRA goes to
 * the Aircraft with the lowest tailNumber.
 **/
public WRA assignNeedyAircraft( WRA newWRA ) {
   Aircraft aircraft = null;
   ArrayList queue = (ArrayList) awpQueues.get( newWRA.getLabel() );
   if( !queue.isEmpty() ) {
      Collections.sort( (List)queue, wraComp );
      for( Iterator iter = queue.listIterator(); iter.hasNext(); ) {
```

```
            aircraft = (Aircraft)iter.next();
         }
         aircraft = (Aircraft)queue.remove(0);
         if( aircraft.getNumAWP(newWRA) > 1 ) {
            queue.add(aircraft);
         }
      }
      newWRA.setBelongsTo(aircraft);
      return newWRA;
   }

   /**
    * Returns the additional amount of repair time necessary to complete
    * repairs of WRAs that arrived for a given Aircraft during the
    * Aircraft's active repair period.  All WRAs that arrived when the
    * Aircraft was in an active state of repair are removed from the
    * Aircraft's "moreToDo" list and then added to the Aircraft's
    * "inWork" list.
    **/
   public double transferMoreToDo( Aircraft aircraft ) {
      WRA wra;
      String label;
      double additionalTTR = 0.0;
      while( aircraft.hasMoreToDo() ) {
         wra = aircraft.removeNextToDo();
         aircraft.addInWork(wra);
         additionalTTR += getNextTTR(wra);
      }
      return additionalTTR;
   }

   /**
    * Sets the squadron up at the beginning of the simulation.  Each
    * Aircraft in the squadron is added to the queue of ready Aircraft.
    **/
   public void setReady( Aircraft[] squadron ) {
      for( int i = 0; i < squadron.length; i++ ) {
         addToReady(squadron[i]);
      }
      numStandBy = ready.size();
      numFMC = ready.size();
   }

   /**
    * Adds an Aircraft to the queue of ready Aircraft.  Will not
    * add the Aircraft to the queue if it is already contained
    * in the queue.
    **/
   public void addToReady( Aircraft aircraft ) {
      if( !ready.contains( aircraft ) ) {
         ready.add( aircraft );
      }
   }

   /**
    * Adds an Aircraft to the HashMap of Non-mission Capable (NMC)
    * Aircraft.
```

```
 **/
public void addToNMC( Aircraft aircraft ) {
   aircraft.startNMCTime();
   nmcMap.put( aircraft.toString(), aircraft );
   firePropertyChange("nmcAdd", aircraft);
   firePropertyChange("fmc", numFMC, --numFMC);
}

/**
 * Adds a WRA to the spares kit.  For example, a WRA is added to
 * the spares kit when the WRA arrives from the requisition (OST)
 * process and no Aircraft needs it.
 **/
public void addSpare( WRA wra ) {
   spareInv.addItem( wra );
   firePropertyChange("spareAdd", wra);
}

/**
 * Adds the Aircraft this WRA belongs to, to the appropriate queue of
 * AWP WRAs.  Each WRA has its own AWP queue.  If the Aircraft
 * is pre-existing inside the queue that is being added to, the Aircraft
 * is not added.
 **/
public void addToAWPQueues( WRA failedWRA ) {
   Aircraft aircraft = failedWRA.getBelongsTo();
   ArrayList queue = (ArrayList) awpQueues.get( failedWRA.getLabel() );
   if( !queue.contains(aircraft) ) {
      queue.add( aircraft );
      firePropertyChange( "addToAWPQueue", failedWRA );
   }
}

/**
 * Sets the time to repair (ttr) HashMap of RandomVariates.
 **/
public void setTTR( HashMap ttrMap ) {
   this.ttrMap = ttrMap;
}

/**
 * Sets the ordering and shipping time (ost) HashMap
 * of RandomVariates.
 **/
public void setOST( HashMap ostMap ) {
   this.ostMap = ostMap;
}

/**
 * Sets the spareInv of spare WRA objects.
 **/
public void setSpareInv( WRAInventory spareInv ) {
   this.spareInv = spareInv;
}

/**
 * Each WRA has its own AWP queue.  Each queue contains Aircraft
```

```
 * this WRA is AWP for.
 **/
public void setAWPQueues( HashMap wraMap ) {
   for (Iterator iter = wraMap.keySet().iterator(); iter.hasNext(); ) {
      String key = (String)iter.next();
      ArrayList queue = new ArrayList();
      awpQueues.put(key, queue);
   }
}

/**
 * The actual time an Aircraft is required to conduct a patrol,
 * excluding the transit time to and from the patrol area.
 **/
public void setPatrolTime( double patrolTime )  {
   this.patrolTime = patrolTime;
}

/**
 * The time required for an Aircraft to fly to/from its patrol
 * area (one way).  The transit time will be the same for going to
 * its patrol area as returning from it.
 **/
public void setTransitTime( double transitTime ) {
   this.transitTime = transitTime;
}

/**
 * The duration of an Aircraft's post-flight inspection.
 **/
public void setInspectionTime( double inspectionTime ) {
   this.inspectionTime = inspectionTime;
}

/**
 * Returns the total number of Aircraft.  That is, the total number
 * of FMC and NMC Aircraft.
 **/
public int getTotalNumberOfAircraft() {
   return originalSquadron.length;
}

/**
 * Returns the number of aircraft that are full mission capable
 * (FMC).  FMC Aircraft are defined to be those that are ready,
 * tasked, flying, and being inspected.  Note: Just because an
 * Aircraft is FMC does not mean it is Operationally Available.
 **/
public int getNumFMC() {
   return originalSquadron.length - getNumNMC();
}

/**
 * Returns the number of Aircraft that are NMC.
 **/
public int getNumNMC() {
   return nmcMap.size();
```

```
}

/**
 * Returns the number of Aircraft ready for a sortie.
 **/
public int getNumReady() {
   return ready.size();
}

/**
 * Returns the number of Aircraft that are currently
 * operational (flying).
 **/
public int getNumOperational() {
   return numOperational;
}

/**
 * Returns the number of Aircraft that are currently
 * on stand-by.  That is, the number of Aircraft that
 * are ready for a sortie, but have not taken-off yet.
 **/
public int getNumStandBy() {
   return numStandBy;
}

/**
 * Tells the system that an Aircraft is needed for a sortie.
 **/
public void setNeeded(boolean needed) {
   this.needed = needed;
}

/**
 * Removes the next Aircraft in the queue that are ready for
 * a sortie.  The queue is sorted by class AircraftReadyComparator.
 **/
public Aircraft removeNextReady() {
   Aircraft aircraft = null;

   if( ready.isEmpty() ) {
      throw new RemoveNextReadyException();
   }
   else {
      Collections.sort( (List)ready, readyComp );
      aircraft = (Aircraft)ready.remove(0);
   }
   return aircraft;
}

/**
 * Removes a given Aircraft from the HashMap of NMC
 * Aircraft.
 **/
public void removeFromNMC( Aircraft aircraft ) {
   nmcMap.remove( aircraft.toString() );
   aircraft.endNMCTime();
```

```
   firePropertyChange( "removeNMC", aircraft );
}

/**
 * Returns the patrol time, exclusive of the transit time
 * to/from the patrol area.
 **/
public double getPatrolTime() {
   return patrolTime;
}

/**
 * Returns the transit time.  The flight time to/from the
 * patrol area.
 **/
public double getTransitTime() {
   return transitTime;
}

/**
 * Returns the inspection time.  The duration of an Aircraft's
 * post-flight inspection.
 **/
public double getInspectionTime() {
   return inspectionTime;
}

/**
 * Generates and returns the repair time for a specified WRA.
 **/
public double getNextTTR( WRA wra ) {
   String label = wra.getLabel();
   return ( (RandomVariate)ttrMap.get(label) ).generate();
}

/**
 * Returns the total time of an Aircraft's sortie.  Sortie time
 * is equal to its patrol time plus its transit time to/from
 * the patrol area.
 **/
public double getSortieTime() {
   return getPatrolTime() + 2.0 * getTransitTime();
}

/**
 * Returns true if an Aircraft is needed for a sortie.
 **/
public boolean isNeeded() {
   return needed;
}

/**
 * Returns the number of a particular WRA
 * that are available in the spares kit.
 **/
public int getNumberOfSpare( WRA wra ) {
   return spareInv.getNumberInStock(wra);
```

```
    }

    /**
     * Removes a WRA from the spares kit and returns
     * it.
     **/
    public WRA getSpare( WRA wra ) {
       WRA tempWRA = null;
       if( spareInv.getNumberInStock(wra) > 0 ) {
          tempWRA = spareInv.removeItem( wra );
          firePropertyChange( "spareRemoval", wra );
       }
       return tempWRA;
    }

    /**
     * Resets at the start of each simulation run.
     **/
    public void reset() {
       super.reset();
       ready.clear();
       nmcMap.clear();
       awpQueues.clear();
       spareInv.clear();

       setReady( originalSquadron );
       setAWPQueues( originalWRAMap );
       needed = true;
       numOperational = 0;
       numOperationalFMC = 0;
       numInspect = 0;
    }
}
```

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: Creates an instance of an Aircraft object.
 */

import java.util.*;
import java.text.*;
import simkit.*;
import exceptions.*;

public class Aircraft extends SimEntityBase {

   // instance variables
   private double startSortieTime;     // time the AC takes off for a sortie
   private double totalOperatingTime;  // total operating time of this  AC
   private double startRepairTime;     // time an AC begins a repair process
   private double endRepairTime;       // time the AC ends a repair process
   private double startNMCTime;        // time the AC starts being NMC
   private String tailNumber;          // how each aircraft is identified
   private DecimalFormat fmt;

   protected boolean fmc;              // true if AC is FMC
   protected boolean inFlightFailure;  // true if AC has an
                                       // inFlightWRAFailure
   protected WRAInventory wraInv;      // state variable
   protected WRAInventory awpInv;      // state variable
   protected LinkedList inWorkList;    // state variable
   protected LinkedList failedList;    // state variable
   protected LinkedList moreToDoList;  // state variable

   // constructor methods

   public Aircraft(String tailNumber) {
      setTailNumber( tailNumber );
      wraInv = new WRAInventory();
      awpInv = new WRAInventory();
      inWorkList = new LinkedList();
      failedList = new LinkedList();
      moreToDoList = new LinkedList();
      fmt = new DecimalFormat( "0.00" );
      startSortieTime = Double.NaN;
      startNMCTime = Double.NaN;
   }

   // instance methods

   /**
    * Tail numbers are two digit Strings.  For example,
    * "00", "07", and "11".
    **/
   public void setTailNumber(String tailNumber) {
```

```
   this.tailNumber = tailNumber;
}

/**
 * Sets the time that this Aircraft starts getting repaired.
 **/
public void setStartRepairTime(double startRepairTime) {
   this.startRepairTime = startRepairTime;
}

/**
 * Sets the SimTime this Aircraft will end it's repair.
 **/
public void setEndRepairTime(double endRepairTime) {
   this.endRepairTime = endRepairTime;
}

/**
 * Adds a WRA to the Aircraft.  The WRA that is added is an individual
 * WRA whose instance "belongs" to this Aircraft.  The WRA is setBelongsTo
 * this Aircraft and the WRA's time to failure (ttf) is reset.  The ttf
 * is only reset when added to the Aircraft object.  A WRA should be added
 * to the Aircraft only when initializing the Aircraft or at the
 * conclusion of the WRA's replacement following a repair process.
 **/
public void addWRA(WRA wra) {
   wra.setBelongsTo(this);
   wra.resetTTF();
   wraInv.addItem(wra);
   firePropertyChange( "addWRA", wra );
}

/**
 * Adds multiple copies of a particular WRA to the Aircraft.  This is
 * a method for convienience.
 **/
public void addWRA( WRA wra, int numberOfCopies ) {
   for( int i = 0; i < numberOfCopies; ++i ) {
      addWRA( new WRA( wra ) );
   }
}

/**
 * Adds a WRA to the awaiting parts (awp) WRAInventory.  That is,
 * adding the WRA to this WRAInventory means a spare isn't in the
 * spares kit and/or the part is not available in some way/shape/form.
 * It must be requisitioned (ordered and received) prior to installation.
 **/
public void addAWP( WRA wra ) {
   wra.setBelongsTo(this);
   awpInv.addItem(wra);
   firePropertyChange( "addAWP", wra );
}

/**
 * Adds a WRA to the inWork WRAInventory.  That is, adding the WRA to this
 * WRAInventory means the WRA is being installed on this Aircraft.
```

```
 **/
public void addInWork( WRA wra ) {
   wra.setBelongsTo(this);
   inWorkList.add(wra);
   firePropertyChange( "addInWork", wra );
}

/**
 * If the Aircraft is being repaired when another WRA arrives from the
 * supply process, the inbound WRA is added to this queue so that when
 * the current repair process ends, this queue can be checked to see if
 * more WRAs need to be installed (replaced).  That is, if WRAs are added
 * to this queue, this means that there is more work to do after the
 * Aircraft ends its current active state of repair.
 **/
public void addMoreToDo( WRA wra ) {
   wra.setBelongsTo(this); // just to make sure
   moreToDoList.add( wra );
   firePropertyChange( "addMoreToDo", wra );
}

/**
 * Tells the Aircraft to start a sortie.  All WRAs are notified that a
 * sortie is starting.
 **/
public void startSortie() {
   setInFlightFailure( false );
   Set labels = wraInv.keySet();
   List inventory = null;
   WRA wra;
   for(Iterator iter = labels.iterator(); iter.hasNext();) {
      inventory = wraInv.getWRAList( (String)iter.next() );
      for(Iterator inventoryIter = inventory.listIterator();
         inventoryIter.hasNext();) {
         wra = (WRA)inventoryIter.next();
         wra.startSortie();
         firePropertyChange( "wraStartSortie", wra );
      }
   }
   startSortieTime = Schedule.getSimTime();
}

/**
 * The following method is for testing purposes.  It may be necessary
 * to delete it later.
 **/
public void startSortie( double sortieTime ) {
   Set labels = wraInv.keySet();
   List inventory = null;
   WRA wra;
   double min = Double.POSITIVE_INFINITY;
   double postSortieTTF;
   for(Iterator iter = labels.iterator(); iter.hasNext();) {
      inventory = wraInv.getWRAList( (String)iter.next() );
      for(Iterator inventoryIter = inventory.listIterator();
         inventoryIter.hasNext();) {
         wra = (WRA)inventoryIter.next();
```

```
            postSortieTTF = -(wra.getTTF() - sortieTime);
            if( postSortieTTF > 0.0 && postSortieTTF < min) {
               min = postSortieTTF;
            }
         }
      }
      if( min != Double.POSITIVE_INFINITY ) {
         setInFlightFailure( true );
         waitDelay( "InFlightWRAFailure", min, 2.0 );
      }
   }

   /**
    * Tells the Aircraft to end a sortie.  All WRAs are notified that
    * a sortie is ending.  Each WRA is checked as to if its ttf is < 0.0.
    * If so, it is considered failed and must be added to the Aircraft's
    * failedList.
    **/
   public void endSortie() {
      Set labels = wraInv.keySet();
      List inventory = null;
      WRA wra;
      for(Iterator iter = labels.iterator(); iter.hasNext();) {
         inventory = wraInv.getWRAList( (String)iter.next() );
         for(Iterator inventoryIter = inventory.listIterator();
            inventoryIter.hasNext();) {
            wra = (WRA)inventoryIter.next();
            wra.endSortie();
            firePropertyChange( "wraEndSortie", wra );
            if( wra.isFailed() ) {
               failedList.add( wra );
               inventoryIter.remove();
               firePropertyChange( "failedWRA", wra );
            }
         }
      }
      totalOperatingTime += Schedule.getSimTime() - startSortieTime;
      startSortieTime = Double.NaN;
   }


   /**
    * Sets the time that an in-flight failure (malfunction) occurs.
    **/
   public void setInFlightFailure( boolean inFlightFailure ) {
      this.inFlightFailure = inFlightFailure;
   }

   /**
    * Returns true if there is/was an in-flight failure (malfunction).
    **/
   public boolean getInFlightFailure() {
      return inFlightFailure;
   }

   /**
    * Sets the time the Aircraft becomes non-mission capable (NMC).
```

```
 **/
public void startNMCTime() {
   startNMCTime = Schedule.getSimTime();
}

/**
 * Notifies the Aircraft that it is no longer NMC.
 **/
public void endNMCTime() {
   startNMCTime = Double.NaN;
}

/**
 * Returns the amount of time this Aircraft has been NMC during
 * its current period of being NMC.  Not a sum of all NMC time,
 * but of how much time being NMC since most recently becoming NMC.
 **/
public double getNMCTime() {
   if( isFMC() ) {
      System.out.println( "Possible ERROR in class Aircraft:
   method getNMCTime() accessed, yet Aircraft is FMC." );
   }
   return Schedule.getSimTime() - startNMCTime;
}

/**
 * Returns the cumulative of operating time for this Aircraft.
 * This is reset at the beginning of each simulation run.
 **/
public double getTotalOperatingTime() {
   return totalOperatingTime;
}

/**
 * Protected method.  Updates the Aircrafts FMC status.
 **/
protected void updateFMC() {
    fmc = failedList.isEmpty() && awpInv.isEmpty() &&
   inWorkList.isEmpty() && moreToDoList.isEmpty();
}

/**
 * Returns true if the Aircraft is full mission capable (FMC).
 * False otherwise.
 **/
public boolean isFMC() {
   updateFMC();
   return fmc;
}

/**
 * Returns true if the Aircraft has failed WRAs.
 **/
public boolean hasFailedWRAS() {
   return !failedList.isEmpty();
}
```

```
/**
 * Returns the number of failed WRAs.
 **/
public int getNumFailed() {
   return failedList.size();
}

/**
 * Returns true if the Aircraft has additional WRAs that need to
 * be installed.  WRAs are added via this method when a WRA
 * arrives that the Aircraft needs when the Aircraft is currently
 * in an active state of repair.   After the Aircraft
 * completes the current state of repair, this method is referenced
 * to check if any more WRAs arrived.  If so, this method returns
 * true.
 **/
public boolean hasMoreToDo() {
   return !moreToDoList.isEmpty();
}

/**
 * Returns true if the Aircraft has any WRAs that are in the process
 * of being installed (in work) on this Aircraft.
 **/
public boolean hasInWork() {
   return !inWorkList.isEmpty();
}

/**
 * Returns true if the Aircraft is on the ground.  That is, is not flying.
 **/
public boolean isOnGround() {
   return Double.isNaN( startSortieTime );
}

/**
 * Returns the total quantity of WRAs that this Aircraft is awaiting
 * parts for.
 **/
public int getTotalNumAWP() {
   return awpInv.size();
}

/**
 * Returns the quantity of a type WRA this Aircraft is awaiting
 * parts for.
 **/
public int getNumAWP( WRA wra ) {
   return awpInv.getNumberInStock(wra);
}

/**
 * Returns the quantity of a type WRA this Aircraft is awaiting
 * parts for.
 **/
public int getNumAWP( String label ) {
   return awpInv.getNumberInStock(label);
```

```
}

/**
 * Removes a WRA from the Aircraft's failed WRA list.  The WRA should
 * then be added to the Aircraft as AWP, or inWork.
 **/
public WRA removeFailedWRA() {
   WRA wra = null;
   if( !failedList.isEmpty() ) {
      wra = (WRA)failedList.removeFirst();
   }
   else {
      System.out.print( "Possible ERROR in class Aircraft,
         method removeFailedWRA: " + this );
      System.out.println( " does not have failed WRAs to get." );
   }
   return wra;
}

/**
 * Removes a WRA from awaiting parts (AWP).
 **/
public void removeAWP( WRA wra ) {
   awpInv.removeItem( wra );
   firePropertyChange( "removeAWP", wra );
}

/**
 * Removes a WRA from awaiting parts (AWP).
 **/
public void removeAWP( String label ) {
   WRA wra = awpInv.removeItem( label );
   firePropertyChange( "removeAWP", wra );
}

/**
 * Returns the next WRA in the queue that holds WRAs which arrive
 * to an Aircraft during an active state of repair.
 **/
public WRA removeNextToDo() {
   WRA wra = null;

   try {
      if( moreToDoList.isEmpty() ) {
         throw new RemoveNextToDoException();
      }
      else {
         wra = (WRA)moreToDoList.removeFirst();
         firePropertyChange("removeNextToDo", wra);
      }
   }
   catch(RemoveNextToDoException e) {
      System.out.println(e.getMessage());
      System.exit(0);
   }
   return wra;
}
```

```
/**
 * Returns the tail number as a two digit String.
 * I.e. "00", "07", "11".
 **/
public String getTailNumber() {
   return tailNumber;
}

/**
 * Returns the integer (int) value of the Aircraft's tail number.
 **/
public int getTailNumberInt() {
   return Integer.parseInt(tailNumber);
}

/**
 * Returns the time at which this Aircraft began its current
 * repair session.
 **/
public double getStartRepairTime() {
   return startRepairTime;
}

/*
 * Returns the SimTime this Aircraft will end it's current repair process.
 */
public double getEndRepairTime() {
   return endRepairTime;
}

/**
 * Transfers all WRAs that are currently inWork back to the Aircraft.
 * WRAs returned to the Aircraft via this method are considered to have
 * their installation complete.
 **/
public void transferInWorkToAircraft() {
   WRA wra;
   for(Iterator iter = inWorkList.listIterator(); iter.hasNext();) {
      wra = (WRA)iter.next();
      addWRA(wra);
      iter.remove();
   }
}

/**
 * Resets at the start of each simulation run.
 **/
public void reset() {
   totalOperatingTime = 0.0;
   fmc = true;
   wraInv.clear();
   awpInv.clear();
   inWorkList.clear();
   failedList.clear();
   moreToDoList.clear();
}
```

```
   /**
    * Returns a String of information about this Aircraft.
    **/
   public String paramString() {
      StringBuffer buf = new StringBuffer("\n");
      buf.append(toString());
      buf.append("\nwraInv:");
      buf.append('\n');
      buf.append(wraInv);
      buf.append("\nfailedList:");
      buf.append('\n');
      buf.append( failedList );
      buf.append("\nawpInv:");
      buf.append('\n');
      buf.append( awpInv );
      buf.append("\ninWorkList:");
      buf.append('\n');
      buf.append( inWorkList );
      buf.append("\nmoreToDoList:");
      buf.append('\n');
      buf.append( moreToDoList );
      return buf.toString();
   }

   /**
    * Returns a String with the Aircraft's tail number.
    * For example, "Aircraft 04".
    **/
   public String toString() {
      return "Aircraft " + getTailNumber();
   }
}
```

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: Creates an instance of a weapons replaceable assembly (WRA).
 */

import java.text.DecimalFormat;
import java.util.*;
import simkit.random.*;

import simkit.*;
import simkit.util.*;
import simkit.stat.*;

public class WRA {

   private static int nextID = 0;          // unique ID for each specific WRA

   private DecimalFormat fmt;              // formats numbers
   private String label;                   // how the WRA is referenced
   private String nomenclature;            // name of this WRA
   private Aircraft belongsTo;             // what AC this WRA belongs to
   private RandomVariate nextFailureTime;  // failure time rng
   private double startSortieTime;         // time the WRA starts a sortie
   private double endSortieTime;           // time the WRA ends a sortie
   private int id;                         // unique ID for EACH specific WRA

   protected double ttf;                   // state variable

   /**
    * Three parameter constructor to create an instance of a WRA.
    **/
   public WRA(String label, String nomenclature,
      RandomVariate nextFailureTime) {
      setLabel( label );                        // give the WRA a label
      setNomenclature( nomenclature );          // give the WRA a name
      setNextFailureTime( nextFailureTime );    // failure time rng
      fmt = new DecimalFormat( "0.00" );
      id = ++nextID;
   }

   /**
    * Single parameter constructor to create an instance of a
    * WRA similiar to that of another WRA.
    **/
   public WRA(WRA wra) {
      this( wra.getLabel(), wra.getNomenclature(),
      wra.getNextFailureTime() );
   }

   /**
    * Sets the label of the WRA.
```

```
 **/
public void setLabel( String label ) {
   this.label = label;
}

/**
 * Sets the nomenclature of the WRA.
 **/
public void setNomenclature( String nomenclature ) {
   this.nomenclature = nomenclature;
}

/**
 * Sets the time to failure (ttf) random number generator from which ttf
 * times are created.
 **/
public void setNextFailureTime( RandomVariate nextFailureTime ) {
   this.nextFailureTime = nextFailureTime;
}

/**
 * Generates and resets the WRA's time to failure (ttf).
 **/
public void resetTTF() {
   setTTF( nextFailureTime.generate() );
}

/**
 * Sets the time to failure (ttf).  This method is only accessable
 * internally from this specific WRA object.
 **/
protected void setTTF( double ttf ) {
   this.ttf = ttf;
}

/**
 * Sets this WRA as belonging to a specific Aircraft.
 **/
public void setBelongsTo( Aircraft aircraft ) {
   belongsTo = aircraft;
}

/**
 * Sets the time at which the WRA begins a sortie.
 **/
public void startSortie() {
   startSortieTime = Schedule.getSimTime();
}

/**
 * Re-calculates the WRAs time to failure.  Deducts the time of
 * the sortie from the pre-existing time to failure (ttf).
 **/
public void endSortie() {
   ttf = ttf - Schedule.getSimTime() + startSortieTime;
   startSortieTime = Double.NaN;
}
```

```
/**
 * Returns the time to failure (ttf) random number generator.
 **/
public RandomVariate getNextFailureTime() {
   return nextFailureTime;
}

/**
 * Returns the WRA's time to failure (ttf).
 **/
public double getTTF() {
   return ttf;
}

/**
 * Returns the WRA label. This is usually the 9-digit NIIN.
 **/
public String getLabel() {
   return label;
}

/**
 * Returns the WRA nomenclature (name).
 **/
public String getNomenclature() {
   return nomenclature;
}

/**
 * Returns, but does NOT remove, the Aircraft object
 * that this WRA belongs to.  Returns null if the individual
 * WRA has not been assigned to an Aircraft.
 **/
public Aircraft getBelongsTo() {
   return belongsTo;
}

/**
 * Removes and returns the Aircraft object that this WRA belongs to.
 * Returns null if the individual WRA was never assigned to an
 * Aircraft.
 **/
public Aircraft removeBelongsTo() {
   Aircraft aircraft = getBelongsTo();
   setBelongsTo( null );
   return aircraft;
}

/**
 * Returns true if the WRA is failed.  That is, returns true
 * if the WRA time to failure (ttf) is <= 0.0.
 **/
public boolean isFailed() {
   return ttf <= 0.0;
}
```

```
   /**
    * Returns true if the WRA is on the ground.  That is, returns
    * true if the Aircraft is not flying.
    **/
   public boolean isOnGround() {
      return Double.isNaN( startSortieTime );
   }

   /**
    * Returns a String with information pertaining to this
    * specific WRA.
    **/
   public String toString() {
      StringBuffer buf = new StringBuffer( "ID=" );
      buf.append( id );
      buf.append( ", WRA=" );
      buf.append( getLabel() );
      buf.append( ", " );
      buf.append( getNomenclature() );
      buf.append( ", TTF=" );
      buf.append( fmt.format( getTTF() ) );
      buf.append( ", belongsTo=" );
      buf.append( getBelongsTo() );
      return buf.toString();
   }
}
```

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: Creates an instance of a WRAInventory object.
 *
 * Author:  Arnold Buss
 * Modified By: Capt Michael Margolis
 **/

import java.util.*;
import simkit.*;
import simkit.util.*;
import java.beans.*;

public class WRAInventory extends SimEntityBase {

   // instance variables

    HashMap inventory;     // each WRA has its own inventory

    /**
     * Creates a new instance of WRAInventory.
     **/
    public WRAInventory() {
        inventory = new HashMap();
    }

    /**
     * Adds a Collection of items to the WRA inventory.  Of the
     * items in the Collection, only WRA objects are added.
     **/
    public void addItems(Collection items) {
        for (Iterator i = items.iterator(); i.hasNext(); ) {
            Object item = i.next();
            if (item instanceof WRA) {
                addItem( (WRA) item);
            }
        }
    }

    /**
     * Adds a specified number of copies of a WRA object
     * to the inventory.
     **/
    public void addCopies(WRA item, int numberCopies) {
        for (int i = 0; i < numberCopies; ++i) {
           addItem( new WRA(item) );
        }
    }

    /**
     * Adds a WRA to the inventory.
```

```
 **/
public void addItem(WRA item) {
    String label = item.getLabel();
    List items = (List) inventory.get(label);
    if (items == null) {
        items = new LinkedList();
        inventory.put(label, items);
    }
    items.add(item);
}

/**
 * Removes a WRA with the given nomenlcature from the
 * inventory.  There may be several instances of a WRA
 * with this label and only one of them will be
 * removed.
 **/
public WRA removeItem(String label) {
    WRA item = null;
    List items = (List) inventory.get(label);
    if (items != null && !items.isEmpty()) {
        item = (WRA) items.remove(0);
    }
    return item;
}

/**
 * Removes a WRA similiar to the one specified from
 * the inventory.  The exact instance of WRA specified
 * may not be removed, but one with the same label
 * will.
 **/
public WRA removeItem(WRA item) {
    return removeItem(item.getLabel());
}

/**
 * Returns the number of WRA objects with this label
 * are in the inventory.
 **/
public int getNumberInStock(String label) {
    List items = (List) inventory.get(label);
    int numberItems = 0;
    if (items != null) {
        numberItems = items.size();
    }
    return numberItems;
}

/**
 * Returns the number of WRA objects with this WRA's
 * label are in the inventory.
 **/
public int getNumberInStock(WRA wra) {
   return getNumberInStock(wra.getLabel());
}
```

```
 /**
  * Returns the Set of keys used to identify WRA objects
  * in the inventory.  The keys refer to the WRA
  * labels.
  **/
 public Set keySet() {
    return inventory.keySet();
 }

 /**
  * Returns a List of WRA objects in the inventory that have the
  * same label.  For example, if there are 5 widgets in
  * the inventory, a List of all 5 widgets will be returned.
  **/
 public List getWRAList(String label) {
    return (List) inventory.get(label);
 }

 /**
  * Returns the total number of WRA objects in the inventory.
  **/
 public int size() {
    int count = 0;
    for( Iterator iter = inventory.keySet().iterator();
       iter.hasNext(); ) {
       count += getNumberInStock( (String)iter.next() );
    }
    return count;
 }

 /**
  * Returns true if the total number of WRA objects in inventory
  * is zero.
  **/
 public boolean isEmpty() {
   boolean empty = false;
   if( size() == 0 ) {
      empty = true;
   }
   return empty;
 }

 /**
  * Clears the inventory.
  **/
 public void clear() {
    inventory.clear();
 }

/**
 * Returns a String listing the entire inventory.
 **/
 public String toString() {
     StringBuffer buf = new StringBuffer("WRA Inventory");
     buf.append('\n');
     buf.append(" total size=");
     buf.append(size());
```

```
        for (Iterator i = inventory.keySet().iterator(); i.hasNext(); ) {
            String label = i.next().toString();
            buf.append('\n');
            buf.append(label);
            buf.append(" - ");
            buf.append(getNumberInStock(label));
            buf.append(" items:");
            List items = (List) inventory.get(label);
            if (items != null) {
                for (Iterator j = items.iterator(); j.hasNext(); ) {
                    buf.append("\n\t");
                    buf.append(j.next());
                }
            }
        }
        return buf.toString();
    }
}
```

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: Creates an instance of an AircraftReadyComparator object.
 **/

import java.util.*;

public class AircraftReadyComparator implements Comparator {

   // instance methods

   /**
    * The sorting criterea for ready Aircraft.
    * Priority:
    *    1) lowest totalOperatingTime
    *    2) lowest tailNumber
    **/
   public int compare(Object first, Object second) {
      if (first == second) { return 0; }
      if (first instanceof Aircraft && second instanceof Aircraft) {
         Aircraft f = (Aircraft) first;
         Aircraft s = (Aircraft) second;

         /*
          * Sort according to lowest totalOperatingTime.
          * That is, smaller totalOperatingTime = higher priority.
          */

         if (f.getTotalOperatingTime() < s.getTotalOperatingTime())
            { return -1; }
         if (f.getTotalOperatingTime() > s.getTotalOperatingTime())
            { return 1; }

         /*
          * Just in case there is a tie for totalOperatingTime, sort
          * according to tailNumber.  That is, smaller
          * tailNumber = higher priority.
          */

         if (f.getTailNumberInt() < s.getTailNumberInt())
            { return -1; }
         if (f.getTailNumberInt() > s.getTailNumberInt())
            { return 1; }
         return 0;
      }
      else {
         throw new IllegalArgumentException("Not an aircraft!");
      }
   }
}
```

```
package ace;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 * Comments: Creates an instance of an AircraftWRAComparator object.
 *
 * Author: Arnold Buss
 * Modified By: Capt Michael Margolis
 **/


import java.util.*;

public class AircraftWRAComparator implements Comparator {

   // instance methods

   /**
    * The sorting criterea for WRAs.
    * Priority:
    *    1) total number of AWP
    *    2) most NMC time
    *    3) lowest tail number
    **/
   public int compare(Object first, Object second) {
      if (first == second) { return 0; }
      if (first instanceof Aircraft && second instanceof Aircraft) {
         Aircraft f = (Aircraft) first;
         Aircraft s = (Aircraft) second;

         /*
          * Sort according to lowest total # of AWP WRAs.
          * That is, smaller total # AWP = higher priority.
          */

         if (f.getTotalNumAWP() < s.getTotalNumAWP()) { return -1; }
         if (f.getTotalNumAWP() > s.getTotalNumAWP()) { return 1; }


         /*
          * Sort according to most amount of time NMC.
          * That is, bigger NMC time = higher priority.
          */

         if (f.getNMCTime() < s.getNMCTime()) { return 1; }
         if (f.getNMCTime() > s.getNMCTime()) { return -1; }

         /*
          * Just in case the above two are both tied,
          * sort according to tail number.
          */

         if (f.getTailNumberInt() < s.getTailNumberInt()) { return -1; }
```

```
            if (f.getTailNumberInt() > s.getTailNumberInt()) { return 1; }
            return 0;
        }
        else {
            throw new IllegalArgumentException("Not an aircraft!");
        }
    }
}
```

```
package exceptions;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 */

public class RemoveNextReadyException extends RuntimeException {

   public RemoveNextReadyException() {
      super("Attempted removal from empty queue of ready Aircraft");
   }

   public RemoveNextReadyException(String message) {
      super(message);
   }
}
```

```
package exceptions;

/**
 * Capt Michael Margolis <BR>
 * SIMACE v. 6.0 <BR>
 * Thesis Project <BR>
 * July 1, 2003 <BR>
 * <P>
 */

public class RemoveNextToDoException extends RuntimeException {

   public RemoveNextToDoException() {
      super("No more WRAs to repair.");
   }

   public RemoveNextToDoException(String message) {
      super(message);
   }
}
```

LIST OF REFERENCES

Blanchard, B. S., Logistics Engineering and Management, 5th ed., Prentice Hall, 1998.

Burrows, J. and Gardner, J., Personal Computer Aviation Retail Requirements Oriented to Weapon Replaceable Assemblies (ARROWs) Version 1.0 Users Manual, Navy Ships Parts Control Center, Mechanicsburg, Pennsylvania, March 1994.

Buss, A. and Stork, K., SIMKIT Version 1.2.6, Naval Postgraduate School, 2002.

Department of Defense (DoD) Instruction 4140.1-R, DoD Material Management Regulation, May 1998.

Elward, Brad, "Lockheed Martin P-3 Orion US Service," World Airpower Journal, Volume 43, pp. 49-108, Winter 2000.

Fabrycky, W. and Blanchard, B., Systems Engineering and Analysis, 3d ed., Prentice Hall, 1998.

Hunter, David, Beginning XML, 2nd ed., Wrox Press Ltd, pp. 15-18, 2001.

Kim, Larry, The Official XMLSPY Handbook, Wiley Publishing, Inc., p. 88, 2003.

Law, A. and Kelton, W., Simulation, Modeling, and Analysis, 3rd ed., McGraw Hill, 2000.

Locks, Mitchell, O., Reliability, Maintainability, and Availability Assessment, Hydan Book Company, Inc., p. 5, 1973.

Naval Aviation Logistics Data Analysis (NALDA), [https://www.nalda.navy.mil], 9 January 2003.

Office of the Chief of Naval Operations Instruction 3000.12 (OPNAVINST 3000.12), Operational Availability of Equipments and Weapons Systems, p. 2, 29 December 1987.

Office of the Chief of Naval Operations Instruction 4790.2H (OPNAVINST 4790.2H), Naval Aviation Maintenance Program (NAMP), Vol. I, pp. C19-C35, 1 June 2001.

Ray, Erik, T., Learning XML, O'Reilly & Associates, Inc, pp. 2-3, 2001.

Sherbrooke, Craig, C., Optimal Inventory Modeling of Systems, Multi-Echelon Techniques, John Wiley & Sons, Inc., p. 32, 1992.

Werenskjold, G. K., An Exploratory Analysis of Corrective Maintenance During Extended Surface Ship Deployments, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

7. Director, Studies and Analysis Division, MCCDC, Code C45
   Quantico, Virginia

8. Professor Arnold Buss
   Modeling, Virtual Environments and Simulation Institute
   Naval Postgraduate School
   Monterey, California

9. Distinguished Professor David Schrady
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

10. Professor Gordon Bradley
    Department of Operations Research
    Naval Postgraduate School
    Monterey, California

11. Mr. Jack Keane
    Johns Hopkins University Applied Physics Laboratory
    Laurel, Maryland

12. Mr. Bill Kroshl
Johns Hopkins University Applied Physics Laboratory
Laurel, Maryland

13 Mr. Rich Miller
Johns Hopkins University Applied Physics Laboratory
Laurel, Maryland

14. LtCol Gregory K. Mislick
Department of Operations Research
Naval Postgraduate School
Monterey, California

15. Captain Michael Margolis
Reston, Virginia